
Code Version 2.4

Center for Space Environment Modeling
The University of Michigan

July 17, 2018
This code is a copyright protected software. (c) 2002- University of Michigan
# Contents

1 Introduction .................................................. 7
   1.1 Acknowledgments ........................................... 8
   1.2 The SWMF in a Few Paragraphs ............................ 9
   1.3 System Requirements .................................... 11

2 Quick Start ..................................................... 13
   2.1 A Brief Description of the SWMF Distribution .......... 13
   2.2 General Hints ............................................. 13
   2.3 Installation ................................................. 14
   2.4 Platform specific information ............................ 18
   2.5 Building and Running an Executable .................... 18
   2.6 Post-Processing the Output Files ....................... 20
   2.7 Restarting a Run ........................................... 21
   2.8 What’s next? ................................................. 22

3 The Basics ...................................................... 23
   3.1 Configuration of SWMF .................................... 23
      3.1.1 Scripts/Configure.pl ................................. 23
      3.1.2 Selecting physics models with Config.pl ........... 24
      3.1.3 Clone Components .................................. 24
      3.1.4 Registering components with LAYOUT.in ........... 25
      3.1.5 Switching models on and off with PARAM.in ....... 25
      3.1.6 Setting compiler flags ............................... 26
      3.1.7 Configuration of individual components .......... 26
      3.1.8 Using stubs for all components .................... 26
   3.2 PARAM.in .................................................. 27
      3.2.1 Included Files, #INCLUDE ......................... 28
      3.2.2 Commands, Parameters, and Comments .............. 28
      3.2.3 Sessions .............................................. 29
      3.2.4 The Order of Commands .............................. 29
      3.2.5 Iterations, Time Steps and Time ................... 32
   3.3 Execution and Coupling Control ........................ 36
      3.3.1 Processor Layout: LAYOUT.in ....................... 36
      3.3.2 Steady State vs. Time Accurate Execution .......... 38
      3.3.3 Coupling order ...................................... 40
4 Complete List of Input Commands 43

4.1 Input Commands for the Control Module 44
4.1.1 General commands 45
4.1.2 Time and session control 46
4.1.3 Testing and timing 48
4.1.4 Component control 49
4.1.5 Coupling control 51
4.1.6 Restart control 54
4.1.7 Output control 54
4.1.8 Solar coordinate commands 56
4.1.9 Planet commands 57
4.1.10 Stub components 60

4.2 Input Commands for the BATSRUS: GM, EE, SC, IH and OH Components 59
4.2.1 Stand alone mode 59
4.2.2 Planet parameters 60
4.2.3 User defined input 62
4.2.4 Testing and timing 65
4.2.5 Initial and boundary conditions 67
4.2.6 Grid geometry 78
4.2.7 Initial time 83
4.2.8 Time integration 83
4.2.9 Implicit parameters 87
4.2.10 Stopping criteria 91
4.2.11 Output parameters 91
4.2.12 Amr parameters 93
4.2.13 Scheme parameters 126
4.2.14 Coupling parameters 135
4.2.15 Physics parameters 139
4.2.16 Corona specific commands 153
4.2.17 Heliosphere specific commands 158
4.2.18 Wave specific commands 159
4.2.19 Script commands 160

4.3 Input Commands for the Ridley Ionosphere Model: IE Component 161
4.3.1 Testing 161
4.3.2 Input and output 161
4.3.3 Physical parameters 162
4.3.4 Scheme parameters 166

4.4 Input Commands for the CIMI: IM Component 167
4.4.1 Numerical scheme 167
4.4.2 Input/output 167
4.4.3 Physics parameters 168

4.5 Input Commands for the CRCM: IM Component 170
4.5.1 Numerical scheme 170
4.5.2 Input/output 170

4.6 Input Commands for the Hot Electron Ion Drift Integrator: IM Component 172
4.6.1 Testing 172

4.7 Input Commands for the Rice Convection Model 2: IM Component 178
4.7.1 Testing 178
4.7.2 Output 178
4.7.3 Time integration 179
4.7.4 Physics parameters 179
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. 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 CAAA 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 Oehrke, 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 CRCM model was developed by Mei-Ching Fok, Natasha Buzulukova and Alex Glocer at the NASA Goddard Space Flight Center. 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. RAM-SCB is currently only available at Los Alamos National Laboratory. The CRCM, HEIDI and RAM-SCB models are also 2 dimensional in space, but they resolve energy as well as pitch angle.

- The Particle-in-Cell (PC) model is the implicit PIC code iPIC3D 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++.

- 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.

- One of the Solar Energetic Particle (SP) models is the Kőta’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 Field Line AdvectionFLAMPA Model for Particle Acceleration (FLAMPA) by Igor Sokolov. FLAMPA 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 Gibson-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 atmosphere 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
CHAPTER 1. INTRODUCTION

- 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 \texttt{lib}, which is created as part of the installation process described later in this document. The executable \texttt{SWMF.exe} is created in the directory \texttt{bin}, which is created during the compilation. If a user does not want to use some particular component, this component should be configured 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) are defined in the \texttt{LAYOUT.in} file. The input parameters are read from the \texttt{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 between 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 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 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, gfortran, ifort, Lahey, NAG, PGF90, and XLF90. The list of C compilers include gcc, clang, icc, pgcc, 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) or VisIt. 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** - description of CON parameters, parameter and layout 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** - a short instruction on installation and usage
- **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 with 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
```
CHAPTER 2. QUICK START

**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/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 file. To keep this user manual more up-to-date and consistent, the README file is quoted verbatim below.

```bash
# 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 source code from CVS (UM only) or from the tar balls.

# The minimum requirement is the SWMF repository.

# You may also need the SWMF_data repository that contains large data files.
# At the UofM the SWMF_data may already be available through a shared disk as /csem1/SWMF_data

# If it is not available through the shared disk, the SWMF_data repository can be put into the home directory or into the SWMF directory.

# Some data files used by the Center for Radiative Shock Hydrodynamics (CRASH) are in the CRASH_data repository. On many machines at the UofM this is already available through a shared disk as /csem1/CRASH_data

# Otherwise it has to be placed into the home directory.

# !!!! Getting SWMF from CVS (requires CVS access) !!!!
# Set the CVS environment variables (with your user name)

setenv CVSROOT Your_User_Name_On_Herot@herot.engin.umich.edu:/CVS/FRAMEWORK
setenv CVS_RSH ssh

# Check out the SWMF distribution
cd {where_you_want_to_have_the_swmf}
cvs checkout SWMF

# Check out the SWMF_data distribution into the home directory if needed
cd
cvs checkout SWMF_data

# Check out the CRASH_data distribution into the home directory if needed
cd
cvs checkout CRASH_data

# !!!! Opening tar balls !!!!

# Open SWMF tar ball
cd {where_you_want_to_have_the_swmf}
tar -xzf {path_to_file}/SWMF_v{version_number}.tgz
# If needed, open the SWMF_data tar ball into the home directory

```bash
cd
tar -xzf {path_to_file}/SWMF_data.tgz
```

# If needed, open the CRASH_data tar ball into the home directory

```bash
cd
tar -xzf {path_to_file}/CRASH_data.tgz
```

```
###############
# 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 Fortan compiler should be explicitly given.
# To see available choices, type

Config.pl -compiler

# Then install the code with the selected Fortran compiler, e.g.

Config.pl -install -compiler=ifort

# 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.

# 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 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 and ps2pdf be installed on your system.

# To create the PDF manuals type
make PDF

# in the SWMF directory. The manuals will be in the doc/ directory.

# Cleaning the documentation
cd doc/Tex
make clean

# To remove all the created documentation type
cd doc/Tex
make cleanpdf

#############################
# 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 mpirun 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 MIPRUN='mpirun -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 test3 on a single core use
make -j test3 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 components. 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:8,8,8,400,100

command sets the block size for the GM component to $8 \times 8 \times 8$ cells, the maximum number of blocks per processor to 400, and the maximum number of implicit blocks per processor to 100. 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 -00

to debug the code with 0 optimization level, and
2.5. BUILDING AND RUNNING AN EXECUTABLE

```
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 post processing codes for BATSRUS can be compiled with

```
make PSPH
make PIDL
```
These two commands will create the codes `bin/PostSPH.exe`, for post processing spherical Tecplot files, and `bin/PostIDL.exe` for post processing IDL files.

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 subdirectories 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 two input files: `LAYOUT.in` and `PARAM.in`. The `LAYOUT.in` file defines the processor layout for the components involved in the future run. 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 file `LAYOUT.in` to run the executable with five components on 16 processors is:
CHAPTER 2. QUICK START

The file syntax is simple. It must start with the directive #COMPONENTMAP and end with another directive #END. Each line between these directives specifies the label for component, i.e. IE, GM and etc., its first and last processor, all relatively 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 and LAYOUT.in files 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
mpirun -np 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
2.7. RESTARTING A RUN

```
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 (e.g. 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 (compiled) components can be registered to participate in the run in the LAYOUT.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 \texttt{-v} flag of the \texttt{Config.pl} script, which edits the Makefile.def file. For example

\begin{verbatim}
Config.pl -v=GM/BATSRUS,IM/RCM2,IE/Ridley_serial
\end{verbatim}

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

\begin{verbatim}
Config.pl -l
\end{verbatim}

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 \texttt{-v} flag of the Config.pl script. For example the Empty UA component can be selected with

\begin{verbatim}
Config.pl -v=UA/Empty
\end{verbatim}

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

\begin{verbatim}
Config.pl -v=Empty,GM/BATSRUS,IE/Ridley_serial
\end{verbatim}

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 \textit{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 \texttt{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

\begin{verbatim}
Config.pl -v=IH/BATSRUS
\end{verbatim}

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 repository, type

\begin{verbatim}
make cleanc clones
Config.pl
\end{verbatim}

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

\begin{verbatim}
Config.pl -uninstall
\end{verbatim}

command.
3.1.4 Registering components with LAYOUT.in

The components used in particular run has to be listed (registered) in the LAYOUT.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 LAYOUT.in file:

```
ID first last stride
#COMPONENTMAP
IE 0 1 1
GM 2 9999 1
#END
```

The first column contains the component ID, the second is the index of the first (root) processor for the component, the third column is the last processor and the last column contains the stride that is typically set to 1. In the example above IE will run on the first 2 PE-s, while GM will run on the rest of the available PE-s. Changing the LAYOUT.in file to:

```
ID first last stride
#COMPONENTMAP
GM 0 999 1
#END
```

will still use the same executable, but will not allow the IE physics component to participate in the run.

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 CFGLAS 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/BATS RUS code, for example, can be configured to use specific equation and user modules. For example

```
cd GM/BATS RUS
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
```

The grid sizes of the various components can be set with the `-g` flag of the `Config.pl` script. For example the

```
Config.pl -g=UA: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 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, together with the LAYOUT.in 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

```
Param/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

```plaintext
#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 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:

```plaintext
#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
```
3.2. PARAM.IN

-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
3.2. PARAM.IN

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/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,

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

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

```plaintext
#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 \texttt{nITER}, \texttt{n_step} and \texttt{time_simulation} at which restart files will be written in each session.
==SESSION 1

#TIMEACCURATE
F DoTimeAccurate

#SAVERESTART
T DoSaveRestart
200 DnSaveRestart
-1.0 DtSaveRestart

#STOP
400 MaxIteration
-1. tSimulationMax

#RUN ==END OF SESSION 1==

#SAVERESTART
T DoSaveRestart
300 DnSaveRestart
-1.0 DtSaveRestart

#STOP
1000 MaxIteration
-1. tSimulationMax

#RUN ==END OF SESSION 2==

#TIMEACCURATE
T DoTimeAccurate

#SAVERESTART
T DoSaveRestart
3 1100 1100 100.0
-1 DnSaveRestart
100.0 DtSaveRestart

#STOP
-1 MaxIteration
300.0 tSimulationMax

#RUN ==END OF SESSION 3==

#SAVERESTART
T DoSaveRestart
4 1400 1400 400.0
-1 DnSaveRestart
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 \texttt{#STOP} command are cumulative, the \texttt{#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 \texttt{#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.

<table>
<thead>
<tr>
<th>Session</th>
<th>nITER</th>
<th>nStep</th>
<th>time_simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>200</td>
<td>1000.0</td>
</tr>
<tr>
<td>2</td>
<td>700</td>
<td>2700</td>
<td>1500.0</td>
</tr>
<tr>
<td>2</td>
<td>1000</td>
<td>3000</td>
<td>2000.0</td>
</tr>
</tbody>
</table>

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

\texttt{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 component layout file LAYOUT.in and the input parameter file PARAM.in. Defining the most efficient 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: LAYOUT.in

Within one run the layout is determined by the #COMPONENTMAP command in the LAYOUT.in file. The exact syntax of this file is documented in the reference manual of the CON_world class. 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.in file should be the following

\[
\begin{align*}
\text{ID} & \quad \text{proc0 last stride} \\
\text{#COMPONENTMAP} \\
\text{GM} & \quad 0 \quad 999 \quad 1 \\
\text{#END}
\end{align*}
\]

If the SWMF is run on less than 999 processors (which is likely), the rank of the last processor will be reduced to the maximum rank of the processors, 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

\[
\begin{align*}
\text{ID} & \quad \text{proc0 last stride} \\
\text{#COMPONENTMAP} \\
\text{IM} & \quad 0 \quad 0 \quad 1 \\
\text{GM} & \quad 0 \quad 999 \quad 1 \\
\text{#END}
\end{align*}
\]

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:
3.3. EXECUTION AND COUPLING CONTROL

ID proc0 last stride
#COMPONENTMAP
IM 0 0 1
GM 1 999 1
#END

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 0 999 1
SC 0 999 1
#END

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 0 999 2
SC 0 999 2
#END

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.
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
```

means that GM 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

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 0 1 1
UA 2 33 1
GM 34 999 1
#END
```

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:
convergence rate to steady state. The default frequency is \( \text{DnRun}=1 \), 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

\[
\text{#COUPLE2} \\
\text{IM} \quad \text{NameComp1} \\
\text{GM} \quad \text{NameComp2} \\
10 \quad \text{DnCouple} \\
-1. \quad \text{DtCouple}
\]

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

\[
\text{! No need to run SC too often, it is already in steady state} \\
\text{#CYCLE} \\
\text{SC} \quad \text{NameComp} \\
1000 \quad \text{DnRun}
\]

\[
\text{! No need to couple SC to IH too often} \\
\text{#COUPLE1} \\
\text{SC} \quad \text{NameSource} \\
\text{IH} \quad \text{NameTarget} \\
1000 \quad \text{DnCouple} \\
-1.0 \quad \text{DtCouple}
\]

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

\[
\text{#TIMEACCURATE} \\
\text{T} \quad \text{DoTimeAccurate}
\]

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
CHAPTER 3. THE BASICS

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

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 time 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

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

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

#COUPLE2

GM NameComp1
IE NameComp2
10.0 DtCouple
-1 DnCouple
3.3. EXECUTION AND COUPLING CONTROL

#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/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
  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.
4.1. INPUT COMMANDS FOR THE CONTROL MODULE

#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

DoTimeAccurate

If DoTimeAccurate is set to true, the SWMF is solving a time dependent problem. If DoTimeAccurate 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.

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
3600.0 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. 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. The default value is nStep=0.

#STOP command

#STOP
100 MaxIteration
10.0 tSimulationMax [sec]

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. The default values are MaxIteration=0 and tSimulationMax = 0.0, so 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.

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.

#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 LAYOUT.in file.

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

3600.0 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

#TEST command

#TEST
CON_session::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         Version

The Version variable contains the version number of SWMF. This command serves to check consistency for restarted runs. The #VERSION command and the Version variable are saved into the restart file.

There is no default value. If the command is not used, the version number is not checked.

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.
#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 registered in the LAYOUT.in file 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_IH 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) equals 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.

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=false, 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

SMF_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

```plaintext
#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

```plaintext
#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

```plaintext
#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.

### #STDOUT command

```plaintext
#STDOUT
F UseStdout
```

The UseStdout variable determines whether the output should go to the standard output or into the log files. This can be set in the #STDOUT command.

The default is to go to the standard output, i.e. UseStdout=T.
4.1. INPUT COMMANDS FOR THE CONTROL MODULE

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

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]
0.00000199 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-4 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.

4.1.10 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

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

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

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

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

This command is only used in stand alone mode.

If DoTimeAccurate is set to true, BATSRUS solves a time dependent problem. If DoTimeAccurate 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

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 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 BATRSUS 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 BATRSUS 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**

<table>
<thead>
<tr>
<th>T</th>
<th>IsRotAxisPrimary (rest of parameters read if true)</th>
</tr>
</thead>
<tbody>
<tr>
<td>23.5</td>
<td>RotAxisTheta</td>
</tr>
<tr>
<td>198.3</td>
<td>RotAxisPhi</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>T</th>
<th>UseRotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>24.06575</td>
<td>RotationPeriod [hour] (read if UseRotation is true)</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>T</th>
<th>IsMagAxisPrimary (rest of parameters read if true)</th>
</tr>
</thead>
<tbody>
<tr>
<td>34.5</td>
<td>MagAxisTheta [degree]</td>
</tr>
<tr>
<td>0.0</td>
<td>MagAxisPhi [degree]</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>MagCenterX</th>
<th>MagCenterY</th>
<th>MagCenterZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>-0.02</td>
<td>0.0</td>
</tr>
</tbody>
</table>

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 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 the commands:

#ROTATIONAXIS
T IsRotAxisPrimary
0.0 RotAxisTheta
0.0 RotAxisPhi

#MAGNETICAXIS
F IsMagAxisPrimary

but much shorter.

4.2.3 User defined input

#USERFLAGS command

#USERFLAGS
F UseUserInnerBcs
F UseUserSource
F UseUserPerturbation
F UseUserOuterBcs
CHAPTER 4. COMPLETE LIST OF INPUT COMMANDS

F UseUserICs
F UseUserSpecifyRefinement
F UseUserLogFiles
F UseUserWritePlot
F UseUserAMR
F UseUserEchoInput
F UseUserB0
F UseUserInitSession
F UseUserUpdateStates

This command controls the use of user defined routines in ModUser.f90. For each flag that is set, an associated routine will be called in the user module. Default is .false. for all flags.

### #USERINPUTBEGIN command

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

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

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.

### #TEST command

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

Examples:
read_inputs - echo the input parameters following the #TEST line
project - 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.
4.2. INPUT COMMANDS FOR THE BATSRUS: GM, EE, SC, IH AND OH COMPONENTS

#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. If both are set, the final one in the session will set the test point.

#TESTXYZ command

#TESTXYZ
1.5 xTest (X coordinate of cell for testing)
-10.5 yTest (Y coordinate of cell for testing)
-10. zTest (Z coordinate of cell for testing)

The location of test info in terms of coordinates. Note that the user should set #TESTIJK or #TESTXYZ, not both. If both are set, the final one in the session will set the test point.

#TESTVAR command

#TESTVAR
12 NameTestVar

p NameTestVar

Index or the name of the variable to be tested. The name should agree with one of the names in the NameVar 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.

#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
CHAPTER 4. COMPLETE LIST OF INPUT COMMANDS

#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

#CODEVERSION command

#CODEVERSION
7.50  CodeVersion

Checks CodeVersion. Prints a WARNING if it differs from the CodeVersion defined in ModMain.f90.

#USERMODULE command

#USERMODULE
TEST PROBLEM Smith
1.3  VersionUserModule

Checks the selected user module. If the name or the version number differs from that of the compiled user
module, a warning is written, and the code stops in strict mode (see #STRICT command). This command
and its parameters are 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 than 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
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 matchcopyrestartvariables in ModRestartFile.f90 for more detail.

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

#SPECIFYRESTARTVARMAPPING command

#SPECIFYRESTARTVARMAPPING
DoSpecifyRestartVarMapping

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

#PRECISION command

#PRECISION
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
nI
nJ
nK

Define the number of grid points in each dimension. 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 the grid size of the restart files. It is now possible to read restart files with a grid size that differs from the grid size the code is compiled with, but strict mode has to be switched off with the #STRICT command. The #CHECKGRIDSIZE command may also be used to enforce a certain grid size.
Checks block size and number of blocks. Stops with an error message, if nI, nJ, or nK differ from those set in ModSize. Also stops if number of blocks exceeds nBLK*numprocs, where nBLK is defined in ModSize and numprocs is the number of processors. This command is used in the restart headerfile to check consistency, and it is also useful to check if the executable is consistent with the requirements of the problem described in the PARAM.in file.

### #BLOCKLEVELSRELOADED command

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

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 dnTiming 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

The default values are shown above.

### 4.2.5 Initial and boundary conditions

### #NORMALIZATION command

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/cm³. 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
4.2. INPUT COMMANDS FOR THE BATSRUS: GM, EE, SC, IH AND OH COMPONENTS

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, 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 Alfvén 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 km/s, and the density unit is amu/cm^3.

If TypeNormalization="SOLARWIND" then the distance unit is the same as for PLANETARY units, but the velocity and density are normalized to the density and the sound speed of the solar wind. 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 or a Tokamak problem). This normalization is only kept for sake of backwards compatibility.

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
PLANE\_TARY 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.
CHAPTER 4. COMPLETE LIST OF INPUT COMMANDS

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 Type Restart InFile

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
4.2. INPUT COMMANDS FOR THE BATSRUS: GM, EE, SC, IH AND OH COMPONENTS

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
- fixedBI - 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)
CHAPTER 4. COMPLETE LIST OF INPUT COMMANDS

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).
4.2. INPUT COMMANDS FOR THE BATSRUS: GM, EE, SC, IH AND OH COMPONENTS

All boundaries can be identified with strings. Some boundaries can also be identified with an index between -3 and 6. Possible identifiers 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 (#LIMITRA DIUS)
- `coord2max, 4` - max 2nd (gen. coord.) cell boundary (#GRIDGEOMETRY LIMIT)
- `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. The timestep inside solid body is set to SolidLimitDt. Default is UseSolidState=.false.

**#OUTFLOWPRESSURE command**

```
#OUTFLOWPRESSURE

T UseOutflowPressure

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

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

**#INNERBOUNDARY command**

```
#INNERBOUNDARY

ionosphere TypeBcBody

ionosphere TypeBcBody2 !read only if UseBody2=T
```
TypeBcBody determines the boundary conditions at the spherical surface of the inner body when these are described with face boundary conditions. For Cartesian grids this is always the case, because the spherical surface is not aligned with the grid blocks, so a ghost cell based boundary condition is not possible. For spherical grids, however, both the cell and face based boundary conditions can be used depending on the combination of commands. If face based boundary is used then the boundary condition at the body surface is determined here as TypeBcBody; if cell based boundary is used then the boundary condition at the body surface is determined by the TypeBc1 parameter of the #OUTERBOUNDARY command.

TypeBcBody2 is only read if the second body is used (see the #SECONDBODY command that has to occur BEFORE this command). The second body can be anywhere in the computational domain, so its spherical 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 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
-'ionospherelfloat/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 #POLARWIND 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.
-'user' - user defined

For 'ionosphere' and 'ionospherelfloat' 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', 'ionospherelfloat/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'.

#INNERBCPE2P 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

-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:

\[ F_{O^+} = \alpha S_\parallel^\beta \]  

...where \( F_{O^+} \) is the local upflowing oxygen flux, \( S_\parallel \) is the local field-aligned Poynting flux, and \( \alpha \) and \( \beta \) are fitting coefficients based on observations from the FAST spacecraft. Default values for \( \alpha \) and \( \beta \), 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 parameter sets the radial velocity of the outflow, which also controls how flux is converted into number density:

\[ n_{O^+} = F_{O^+}/U_R \]  

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.


#MAGNETICINNERBOUNDARY command

T DoReflectInnerB1

If DoReflectInnerB1 is true, the B1 part of the magnetic field is reflected for the ionosphere type inner boundary conditions, i.e. the radial component of the total magnetic field is forced to coincide with the B0 value.

The default value is DoReflectInnerB1 false, which makes B1 float.

#BUFFERGRID command

2 nRBuff
90 nPhiBuff
45 nThetaBuff
19.0 rBuffMin
21 rBuffMax
0.0 PhiBuffMin
360.0 PhiBuffMax
-90.0 LatBuffMin
90.0 LatBuffMax
CHAPTER 4. COMPLETE LIST OF INPUT COMMANDS

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 spherical shell should be used (alternatively, 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.

\#EXTRABOUNDARY command

\#EXTRABOUNDARY
TUseExtraBoundary
userTypeExtraBoundary

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.

\#FACEBOUNDARY command

\#FACEBOUNDARY
6MinFaceBoundary
-2MaxFaceBoundary

This command is no longer used. See \#BOXBOUNDARY, \#OUTERBOUNDARY and \#INNERBOUNDARY for more information.

\#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
20000.0 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). 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.
4.2. INPUT COMMANDS FOR THE BATSRUS: GM, EE, SC, IH AND OH COMPONENTS

#CPCPBOUNDRY command

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:

\[ \text{RhoBc} = \text{Rho0Cpcp}[i] + \text{RhoPerCpcp}[i] \times \text{Cpcp} \]

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

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

#YOUNGBOUNDARY command

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

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 \( \text{RhoNeuFactor} \times \text{Rho1}, \text{RhoNeuFactor} \times \text{P1} \) and \( \text{uNeuFactor} \times \text{u1} \), where \( \text{Rho1}, \text{p1}, \text{u1} \) 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

400 MaxBlock
100 MaxBlockImpl

This command can be used in the first session of BATSRUS only.

Set the maximum number of grid blocks per processor and the maximum number of blocks advanced by the part-implicit method (see #IMPLICIT). It is a good idea to set these values to be larger but close to the actual number of blocks used during the run to minimize memory use and improve performance. Note that MaxBlockImpl cannot be more than MaxBlock.

The default values are either the original values shown above or set by the Config.pl -g=.... configuration.

#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!).
4.2. INPUT COMMANDS FOR THE BATSRUS: GM, EE, SC, IH AND OH COMPONENTS

#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 Magnetosphere) 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 simulation (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, because 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 extremely 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

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

#GRIDGEOMETRY

roundcube  
 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 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 rRound1 and remains round at the outer boundary. For this case the transformation does not affect the main diagonals and the maximum stretching is applied along the main axes. 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
4.2. INPUT COMMANDS FOR THE BATSRUS: GM, EE, SC, IH AND OH COMPONENTS

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 minimum 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 cylindrical 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
F UsePoleDiffusion
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 UsePoleDiffusion is true, some numerical diffusion applied between the cells on the opposite side of the axis. This diffusion may smooth out some artifacts that show up when a discontinuity crosses the axis.

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 both UsePoleDiffusion and DoFixAxis.

#COARSEAXIS command

#COARSEAXIS

T UsePoleDiffusion
T UseCoarseAxis
1 nCoarseLayer

The computational cells become very small near the symmetry axis of a spherical or grid. When UsePoleDiffusion is true, some numerical diffusion applied between the cells on the opposite side of the axis. This diffusion may smooth out some artifacts that show up when a discontinuity crosses the axis.

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.

Default is false for both UsePoleDiffusion and 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 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. This is a date and time when both the rotational and the magnetic axes have approximately zero tilt towards the Sun.

#TIMESIMULATION command

#TIMESIMULATION

3600.0 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, which provides human readable information about the restart state.
4.2. INPUT COMMANDS FOR THE BATSRUS: GM, EE, SC, IH AND OH COMPONENTS

In SWMF the command is ignored (SWMF has its own #TIMESIMULATION command). In stand alone mode time simulation is set, but in case of a restart, it gets overwritten by the binary value saved into the .rst binary files.

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.

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.

#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 #UPDATECHEK) 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

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

MinCheckVar
MaxCheckVar
RelativeEps(bx)
AbsoluteEps(bx)
RelativeEps(by)
AbsoluteEps(by)
RelativeEps(bz)
AbsoluteEps(bz)
RelativeEps(p)
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

\[
\frac{\max(\text{abs}(\text{State} - \text{StateOld}))}{(\text{RelativeEps}\times\text{abs(\text{State})} + \text{AbsoluteEps})}
\]

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

UsePointImplicit
BetaPointImplicit (read if UsePointImplicit is true)
IsAsymmetric
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 time-stepping 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**

<table>
<thead>
<tr>
<th>UsePointImplicit</th>
<th>UsePartImplicit</th>
<th>UseFullImplicit</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>UseSemiImplicit</th>
<th>radiation</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>None</td>
</tr>
</tbody>
</table>

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
  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
  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 < DtFixed
- elseif (TypeImplCrit =='r' ) then blocks with rMinBLK < 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
  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.
CHAPTER 4. COMPLETE LIST OF INPUT COMMANDS

#IMPLSTEP command

#IMPLSTEP

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>ImplCoeff</td>
</tr>
<tr>
<td>F</td>
<td>UseBdf2</td>
</tr>
<tr>
<td>F</td>
<td>UseSourceImpl</td>
</tr>
</tbody>
</table>

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 UseBdf2=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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>SemiImplCoeff</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>nOrderImpl</td>
</tr>
<tr>
<td>Rusanov</td>
<td>TypeFluxImpl</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>RejectStepLevel</td>
</tr>
<tr>
<td>0.5</td>
<td>RejectStepFactor</td>
</tr>
<tr>
<td>0.6</td>
<td>ReduceStepLevel</td>
</tr>
<tr>
<td>0.95</td>
<td>ReduceStepFactor</td>
</tr>
<tr>
<td>0.8</td>
<td>IncreaseStepLevel</td>
</tr>
<tr>
<td>1.05</td>
<td>IncreaseStepFactor</td>
</tr>
</tbody>
</table>

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:
4.2. INPUT COMMANDS FOR THE BATSRS: GM, EE, SC, IH AND OH COMPONENTS

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

Assigning \( \text{ReduceStepFactor}=1.0 \) means that the time step is not reduced unless the step is rejected. Assigning \( \text{IncreaseStepFactor}=1.0 \) 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
1.E-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 inverse 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**

<table>
<thead>
<tr>
<th>T</th>
<th>DoPrecond (rest of parameters are read if true)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MBILU</td>
<td>TypePrecond (MBILU, BILU, DILU, GS, BLOCKJACOBI, JACOBI, HYPRE)</td>
</tr>
<tr>
<td>0.5</td>
<td>GustafssonPar (0 to 1, read for MBILU preconditioner only)</td>
</tr>
</tbody>
</table>

**#SEMIPRECOND**

<table>
<thead>
<tr>
<th>T</th>
<th>DoPrecond</th>
</tr>
</thead>
<tbody>
<tr>
<td>HYPRE</td>
<td>TypePrecond</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>#SEMIKRYLOV</th>
<th>GMRES</th>
<th>TypeKrylov (GMRES, BICGSTAB, CG)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.001</td>
<td>ErrorMaxKrylov</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>MaxMatvecKrylov</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>#KRYLOVSIZE</th>
<th>nKrylovVector</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>#SEMIKRYLOVSIZE</th>
<th>nKrylovVector</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>#STOP</th>
<th>MaxIteration</th>
<th>tSimulationMax [sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

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.
4.2.11 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 logfile. 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)
Theta1 Theta2 Status - average of colatitude of fieldline tracing (for testing)
Phi1 Phi2 Status - average of longitude and status of fieldline (for testing)
4.2. INPUT COMMANDS FOR THE BATSRUS: GM, EE, SC, IH AND OH COMPONENTS

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
Theta1pnt Theta2pnt - point value of colatitude of fieldline mapping
Phi1pnt Phi2pnt - 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
2 nSatellite
MHD ray StringSatellite
100 DnOutput
-1. DtOutput [sec]
satellite1.dat NameTrajectoryFile
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' or 'HGC'
OptionalVar = 'ray', 'none', 'step', 'time', and/or '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 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
Theta1, Theta2 - colatitude of mapped field line along B and -B
Phi1, Phi2 - longitude 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 'Theta1 Phi1 Theta2 Phi2 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 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
ray - fieldline tracing variables: theta1 phi1 theta2 phi2 status

More than one OptionalVar strings can be listed. They can be put together in any combination. The default value for OptionalVar is 'step date'.
4.2. INPUT COMMANDS FOR THE BATSRUS: GM, EE, SC, IH AND OH COMPONENTS

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 for example. 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).

The default is nSatellite=0, i.e. no satellite data is saved.

\#STEADYSTATE SATELLITE command

\#STEADYSTATE SATELLITE

\-86400.0 SatelliteTimeStart [sec]
86400.0 SatelliteTimeEnd [sec]
\-3600.0 SatelliteTimeStart [sec]
3600.0 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.

### #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. Official 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

#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
```
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
ascii TypeFileMagGrid (ascii, tec, real4, real8)
SMG TypeCoordMagGrid (GEO, MAG, SMG)
3 nLonMagGrid
3 nLatMagGrid
0. LonMinMagGrid
360. LonMaxMagGrid
-80. LatMinMagGrid
80. LatMaxMagGrid
-1 DnSaveMagGrid
60.0 DtSaveMagGrid
```

This command allows calculating and saving magnetic perturbations on a longitude-latitude grid.

TypeFileMagGrid specifies the file format, which 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 number of grid points in the longitude and latitude directions is given by nLonMagGrid and nLatMagGrid, respectively. The longitudes span from from LonMinMagGrid to LonMaxMagGrid, while the latitudes span from LatMinMagGrid to LatMaxMagGrid. 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.
#SAVEPLOT command

#SAVEPLOT
17 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. DxDxSavePlot ! with smallest grid resolution
y=0 VAR idl StringPlot ! y=0 plane plot with listed variables
-1 DnSavePlot
0.25 DxDxSavePlot ! 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. DxDxSavePlot ! unstructured grid (for IDL plots)
los tbl idl_real4 StringPlot ! line of sight plot using table
-1 DnSavePlot
100. DxDxDxSavePlot
-215. ObsPosX
0. ObsPosY
0. ObsPosZ
0. OffsetAngle
32. rSizeImage
0. xOffset
0. yOffset
3. rOccult
0.5 MuLimbDarkening
300 nPix
AiaXrt NameLosTable
4.2. INPUT COMMANDS FOR THE BATSRS: GM, EE, SC, IH AND OH COMPONENTS

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
1cb 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
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]
rfr tec rwi StringPlot
10 DnSavePlot
-1.0 DtSavePlot
-67.92 ObsPosX
200.40 ObsPosY
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 define 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 terms 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

\textbf{PlotForm} \textbf{PlotArea} \textbf{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 BATSRSUS/SWMF is configured with the -hdf flag. The "IDL" format can be read with the IDL visualization macros (getpict, animate, slice) in BATSRSUS/Idl or with the SpacePy python package. The ASCII "IDL" format can be easily read with any other plotting software.

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)
- 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)
- 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
CHAPTER 4. COMPLETE LIST OF INPUT COMMANDS

**ieb** - field lines traced from a subset of the IE coupled grid

**lcb** - last closed field lines

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 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 define 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 Bix By BIZ e Jx Jy Jz
  - pars: g rbody
- **'raw'** - vars: Conservative_Variables P b1x b1y b1z divb
  - pars: g rbody
- **'ray'** - vars: bx by bz theta1 phi1 theta2 phi2 status blk
  - (if DoMapEquatorRay=F)
- **'m hd'** - vars: Primitive_Variables Jx Jy Jz
  - (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 PrimVarMinR 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
- **'euv'** - vars: euv171 euv195 euv284
  - (PlotArea='los' only)
  - pars: mu
- **'sx r'** - vars: sxr
  - (PlotArea='los' only)
  - pars: mu
- **'tbl'** - vars: listed in the LOS table file
  - (PlotArea='los' only)
  - pars: mu
- **'int'** - vars: 1/B n/B p/B
  - (PlotArea='lcb' only)
  - pars:
- **'nul'** - vars:
  - (PlotArea='lcb' only)
  - pars:

Depending on StringPlot, the following parameters are also read from the parameter file in this order:

- **xminCut...zmaxCut** if PlotArea is 'cut', 'dpl', or 'slc'
- **nRadius nLon** if PlotArea is 'eqr' or 'eqb'
4.2. INPUT COMMANDS FOR THE BATSRUS: GM, EE, SC, IH AND OH COMPONENTS

TypeCoordPlot if PlotArea is 'shl' or 'box'
RadiusMin RadiusMax if PlotArea is 'eqr', 'eqb', or 'shl'
LonMin LonMax if PlotArea is 'eqb' or 'shl'
LatMin LatMax if PlotArea is 'shl'
dRadius, dLon, dLat if PlotArea is 'shl' and associated range is nonzero.
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/los/rfr/lin/eqr/eqb
NameVars if PlotVar is 'var' or 'VAR'
NamePars if PlotVar is 'var' or 'VAR'

The six parameters xMinCut ... zMaxCut define a 3D box in (curvilinear) coordinates: the plotting range.

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. 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-site Extreme Ultraviolet (EUV) and Soft X-Ray (SXR) plots, the same parameters 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.

Additionally, because EUV and SXR plots are configured to read in a response table specific to the EUV or SXR instrument (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  FileType

#LOOKUPTABLE
sxr   NameTable
load   NameCommand
SC/Param/los_Sxt.dat NameFile
ascii  FileType
```

The possible values for NameVars with PlotArea 'los' are listed in subroutine set_plotvar_los in write_plot_los.f90.

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 connecting 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 ASCII in 'tec' and 'tcp' 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

\[ \text{IO2/PlotArea_PlotVar_PlotNumber_TIMESTAMP_PEnumber.extension} \]

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

\[ \text{IO2/PlotArea_PlotVar_PlotNumber_TIMESTAMP.headextension} \]

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:

\[ \text{IO2/los_PlotVar_PlotNumber_TIMESTAMP.extension} \]

where extension is 'dat' for TecPlot and 'out' for IDL file formats.

The shell plot area (‘shl’) 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 output is a single file in IDL or Tecplot format.

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

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

This command allows switching off the radio wave refraction to evaluate how the refraction affects the images obtained with 'rfr' plots (see #SAVEPLOT). Default is switched on (UseNoRefraction=F).

#SAVETECPLT command

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 different 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

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

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 _nTIMESTEP format (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.
4.2. INPUT COMMANDS FOR THE BATSRUS: GM, EE, SC, IH AND OH COMPONENTS

#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.

#PLOTFILENAME command

#PLOTFILENAME
hour NameMaxTimeUnit

For time accurate runs the plot filenames contain an 8-character 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    DoSaveInitial

Save plots and log/satellite files at the beginning 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 Amr parameters

#AMRINITPHYSICS command

#AMRINITPHYSICS
nRefineLevelIC 3

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
cylinder 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
CHAPTER 4. COMPLETE LIST OF INPUT COMMANDS

20.0  Radius
2.0  Taper

#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 command 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) to define where some action (e.g. grid refinement) should be performed.

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.

If 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 NameShape starts with ‘user’, the shape is defined by the subroutine user specification. 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**

```
required="F" required="F"

#GRIDRESOLUTION
2.0 Resolution
initial NameShape

#GRIDLEVEL
3 nLevel
all NameShape

#GRIDLEVEL
4 nLevel
box NameShape
-64.0 xMinBox
-16.0 yMinBox
-16.0 zMinBox
-32.0 xMaxBox
16.0 yMaxBox
0.0 zMaxBox

#GRIDLEVEL
4 nLevel
brick NameShape
-48.0 xPosition
0.0 yPosition
-8.0 zPosition
32.0 xSizeBrick
32.0 ySizeBrick
16.0 zSizeBrick

#GRIDRESOLUTION
1/8 Resolution
shell0 NameShape
3.5 RadiusInner
4.5 Radius

#GRIDRESOLUTION
0.5 Resolution
sphere NameShape
-10.0 xPosition
10.0 yPosition
0.0 zPosition
20.0 Radius
```
#GRIDRESOLUTION
1/8 Resolution
cylinderx NameShape
-30.0 xPosition
0.0 yPosition
0.0 zPosition
60.0 Height
20.0 Radius

#GRIDRESOLUTION
1/8 Resolution
ingz0 rotated NameShape
5.0 Height
20.0 RadiusInner
25.0 Radius
10.0 xRotate
10.0 yRotate
0.0 zRotate

#GRIDRESOLUTION
1/4 Resolution
paraboloidx0 stretched NameShape
30.0 Height
10.0 Radius
12.0 RadiusPerp

#GRIDRESOLUTION
1/8 Resolution
user NameShape

The #GRIDRESOLUTION and #GRIDLEVEL commands allow to set the grid resolution or refinement level, respectively, in a given area.

The Resolution parameter of the #GRIDRESOLUTION 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.

The nLevel parameter is an integer with level 0 meaning no refinement relative to the root block, while level N is a refinement by 2 to the power N.

Note that the #REGION commands in combination with the #AMRCRITERIALEVEL and #AMRCRITERIARESOLUTION commands allow even more flexibility in controlling the grid adaptation, but the initial resolution/level still has to be set by this command.

If NameShape 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, #GRIDRESOLUTION 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 NameShape 'all' refers to the whole computational domain, and it can be used to set the overall minimum resolution.

For other values of NameShape, 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.
4.2. INPUT COMMANDS FOR THE BATSRUS: GM, EE, SC, IH AND OH COMPONENTS

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 DxCellMax 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 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

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

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 to 3)
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: "#AMRCRITERIALLEVEL" 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 #AMRRSOLUTION). 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

The #AMRCRITERIALEVEL or #AMRCRITERIARESOLUTION 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 or #AMRCRITERIARESOLUTION command. #AMRCRITERIARESOLUTION is equivalent with #AMRCRITERIALEVEL but works with cell size (MaxResolution) instead of grid level (MaxLevel). For non-Cartesian grids "cell size" is defined as the maximum length of the cell edges inside a block) instead of grid levels. 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 Type-
Criteria determines its type. TypeCriteria="level" and "dx" 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"). 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 #AMRCRITERIALLEVEL) or MaxResolution (for #AMRCRITERIARESOLUTION). 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 #AMRCRITERIALLEVEL) or MaxResolution (for #AMRCRITERIARESOLUTION) 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). The plus sign before the region name means that the AMR criterion is applied in this region, while the minus sign means that the criterion is not applied in this region. If the first modifier has a negative sign, it is assumed that you want the criterion to be evaluated everywhere except in the region(s) specified with - sign(s).

TypeCriteria can be chosen from the following list:

'dx' - refinement based on (max) cell size in a block
'level' - refinement 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 possibilities:

'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 \( \frac{d|\rho U|/dt}{|\rho U|} \)

'transient B_dot' - relative change of magnetic field \( \frac{d|B|/dt}{|B|} \)

'transient meanUB' - max\(\frac{d|\rho U|/dt}{|\rho U|}\) * max\(\frac{d|B|/dt}{|B|}\)

'transient Rho_2nd_1' - \(\frac{(|d^2\rho U/dx^2| + |d^2\rho U/dy^2| + |d^2\rho U/dz^2|)}{\rho}\)

'transient Rho_2nd_2' - \(\frac{(|d^2\rho U/dx^2 + d^2\rho U/dy2 + d^2\rho U/dz2|)}{\rho}\)

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:

\[
E = \frac{d^2 U}{dx^2} \frac{1}{1} \frac{dU}{dx} \frac{SmallError}{SmallError} + \frac{\sum U}{U} + \text{Epsilon} \frac{dx}{dx} \frac{dx^2}{dx^2}
\]

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 #AMRRRESOLUTION).

4.2.13 Scheme parameters

#SCHEME command

#SCHEME
5 nOrder (1, 2, 4 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 4th order finite volume scheme (FIVOL4) uses the classical 4th order Runge-Kutta scheme for time integration. The spatially 5th order schemes uses the 3rd order Runge-Kutta scheme.

NOTE 1: 4th order scheme is not recommended, and it may be removed in the future! The #SCHEME4 command has more options for it.

NOTE 2: 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 3: the time discretization scheme can be modified with the #TIME STEPPING, #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)
'HLLD' - Miyoshi and Kusano’s HLLE flux
'Roe' - Roe’s approximate Riemann flux (new)
'RoeOld' - Roe’s approximate Riemann flux (old)
4.2. INPUT COMMANDS FOR THE BATSRUS: GM, EE, SC, IH AND OH COMPONENTS

'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 HLL 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 other 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 command 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 largest 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.

#SCHEME4 command

This command has to be used in the first session, because it determines the size of various arrays that are allocated at the beginning of the run. The command reads three parameters used by the 4th order finite volume scheme FIVOL4, which can be activated in any session by setting nOrder=4 in the #SCHEME command.

If UseVolumeIntegral4 is true then the cell averaged variables are converted into cell centered variables and back as needed. This requires an extra ghost cell layer. Currently this option only works for Cartesian grids and with cell based boundary conditions (no inner body).

If UseFaceIntegral4 is true, then the face averaged face values and face fluxes are converted to face centered variables and back as needed. This option can only be used on 2D and 3D Cartesian grids and with cell based boundary conditions (no inner body).

If UseLimiter4 is true then the truly 4th order limiter is used, however this requires an extra ghost cell layer, and the improvement is marginal at best.

If all 3 logicals are true and nOrder is set to 4 in the #SCHEME command then the fully 4th order accurate FIVOL4 scheme is used. This, however, requires extra ghost cell layers and extra face flux calculations. In practice the fully 4th order FIVOL4 scheme is much more expensive but not much more accurate than the simplified scheme with all three logicals set to false.

IMPORTANT: the number of ghost cell layers set with Config.pl -ng=.. has to be sufficient for the selected scheme! The FIVOL4 scheme needs 3 to 5 layers. The minimum 3 layers are needed if both UseVolumeIntegral4 and UseLimiter4 are false. If one of these logicals is true, 4 layers are required. If both are true then the maximum 5 ghost cells layers are needed.

The default is false for all 3 logicals.

#FLATTENING command

This command controls the parameters of Phil Collela’s flattening algorithm used by the 4th order scheme (nOrder=4 in the #SCHEME command). The flattening scheme uses a linear combination of the first order and the 4th order accurate face values. Near strong shocks it switches to the first order scheme, elsewhere it uses the 4th order accurate face value.

If the UseFlattening parameter is true, the flattening scheme is switched on.
The UseDuFlat parameter determines if the velocity gradient is used as a condition for flattening. The "official" algorithm uses this check, in some cases it seems to be better not to use it, and it is not clear if there is any advantage using it.

The FlatDelta parameter determines the threshold for the relative pressure jump. If $p_R$ and $p_L$ are the pressures in the $i+1$ and $i-1$ cells, then the flattening is applied only if $|p_R - p_L|$ is greater than $\text{FlatDelta} \times \min(p_R, p_L)$.

The FlatRatioMin parameter determines the lower threshold for the shock width ratio $= \frac{|p(i+2) - p(i-2)|}{|p_R - p_L|}$. If the ratio is less than FlatRatioMin, no flattening is applied.

The FlatRatioMax parameter determines the upper threshold for the shock width ratio. If the ratio is above FlatRatioMax, the first order scheme is used. If the ratio is between the FlatRatioMin and FlatRatioMax values, the 1st order and 4th order face values are linearly combined with a weight $(\text{Ratio} - \text{FlatRatioMin})/(\text{FlatRatioMax} - \text{FlatRatioMin})$.

In the actual scheme, the flattening parameters are calculated for all dimensions and all cells first, and then the minimum is taken over a 3 point stencil in each dimension, and then a minimum is taken over the dimensions. See Miller and Colella 2005, APJS, 160, 199.

The default values are shown in the example.

### 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.

For the GM component the default is using non-conservative pressure equations within 6 planet radii as long as NamePlanet is not set to NONE (see the #PLANET command). Otherwise, and also for the other components, the default is using the conservative energy equation everywhere.

#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

For the GM component with a planet the default values are nConservCrit = 1 with TypeConservCrit = 'r' and rConserv=6. If there is no planet (PLANET="NONE") or for the EE, SC, IH and OH components, the default is to have no conservative criteria: nConservCrit = 0.
4.2. INPUT COMMANDS FOR THE BATSRUS: GM, EE, SC, IH AND OH COMPONENTS

#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 values are shown for the non-Runge-Kutta time integration schemes. For Runge-Kutta schemes UseUpdateCheck is forced to be false.

#CHECKTIMESTEP command

#CHECKTIMESTEP
T DoCheckTimeStep
2 nCheckTimeStep
1e-6 TimeStepMin

This command is only effective in time accurate mode.

If DoCheckTimeStep is true, then check if time step is smaller than TimeStepMin for nCheckTimeStep consecutive time steps. If it is, save the output files (but not restart) and stop the code.

The current implementation only looks at the time step set before update check or other time step control was applied.

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 the #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 \( D = \min(\text{VarNew}/\text{VarOld}) \) 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 \( D = \max(\text{VarNew}/\text{VarOld}) \) where the maximum 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 accepted, but there is no change in the time step/CFL number.

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.
This command sets thresholds for the relative increase in the control variables in the time step control scheme. The relative increase is defined as $I = \max(\Var_{\text{New}}/\Var_{\text{Old}})$ 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 where 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 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

<table>
<thead>
<tr>
<th>#MULTIION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0001</td>
</tr>
<tr>
<td>1e-10</td>
</tr>
<tr>
<td>T</td>
</tr>
<tr>
<td>3.0</td>
</tr>
<tr>
<td>30.0</td>
</tr>
</tbody>
</table>

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 equation

\[ y^2 + z^2 + x \times ParabolaWidthMultiIon = 0 \]

The defaults are LowDensityRatio=0.0001, LowPressureRatio=1e-10, and DoRestrictMultiIon=false.

#MHDIONS command

<table>
<thead>
<tr>
<th>#MHDIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
</tr>
<tr>
<td>T</td>
</tr>
</tbody>
</table>

This command determines how the total MHD fluid and the individual ion fluids are reconciled after each time step. If both DoAddRho and DoAddRhoU are true, then the density and momentum of individual ion fluids are added up and the MHD solution is overwritten. This is mostly useful for testing purposes, because in this case it is better not to solve for the total fluid at all. This is the default, because it is BETTER not to solve for the total fluid.

If DoAddRho is false then the density obtained by the MHD scheme is used to adjust the densities of the ion fluids. They are scaled such that their sum matches the total fluid.

If DoAddRhoU is false, the same scaling is done for the three components of the ion momenta as long as all ion velocities have the same signs! If the signs are mixed, the total momentum is replaced by the sum of the ion momenta. The procedure is done separately for the X, Y and Z components of the momenta. In real problems this algorithm does not seem to work very well.

In short, do NOT use the total fluid.

Default values are shown.
4.2. INPUT COMMANDS FOR THE BATSRUS: GM, EE, SC, IH AND OH COMPONENTS

#MULTIONSTATE command

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 equivalent with a single-fluid multi-species simulation. This is useful for testing.

When UseSingleIonVelocity is true, the ion velocities are set to the velocity of the total fluid \( u = \frac{\sum_s (\rho_s u_s)}{\sum_s \rho_s} \) as if there was an infinitely strong friction force between the ion fluids.

When UseSingleIonTemperature is true, the ion temperatures are set to the temperature of the total fluid \( k_B T = \frac{\sum_s \rho_s}{\sum_s \rho_s M_s} \) as if there was an infinitely fast energy exchange between the ion fluids.

Default values are false for both parameters.

#COLLISION command

This command is only useful for multi-ion 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 \( \text{CollisionCoefDim} \times \frac{n}{T^{1.5}} \) where \( T \) is the temperature measured in Kelvin, \( n \) is the number density measured in \( /\text{cm}^{-3} \) and the resulting rate is in units of \( 1/\text{s} \). 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 \( B/\sqrt{\rho_1 \rho_2/(\rho_1+\rho_2)} \) where \( B \) is the magnetic field magnitude, and \( \rho_1 \) and \( \rho_2 \) are the densities of the two ion fluids in normalized units. Setting uCutOffDim = -2.0 applies a cut-off velocity based on the total Alven speed \( B/\sqrt{\rho_1+\rho_2} \).

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

\[
(1/TauCutOffDim) \min(\rho_i, \rho_j)(u_j - u_i)^2 / uCutOffDim^2 \text{nP owerCutOff},
\]

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, i.e. neither collision, nor friction are applied.

#MESSAGEPASS command

This is a message passing 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.

#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, i.e. 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

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 subcycling 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 res. change" algorithm for the 2nd order scheme, and the 5th order prolongation for the 5th order scheme.

# LIMITER command

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

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 planetary 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.

### #USEB0 command

```plaintext
#USEB0
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 potential field solution for the corona. B1 is not small relative to B0 in general. The default value depends on the application.

### #USECURLB0 command

```plaintext
#USECURLB0
T UseCurlB0
2.5 rCurrentFreeB0 (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).

The default is UseCurlB0 false.

### #LIGHTSPEED command

```plaintext
#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.

### #HYPERBOLICDIVE command

```plaintext
#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

```plaintext
#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.

**#DIVBSOURCE command**

#DIVBSOURCE
T UseB0Source

Add extra source terms related to the non-zero divergence and curl of B0. Default is true.

**#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':

\[ \| \text{div } B \| < \text{RelativeLimit} \| \text{div } B_0 \| \]

TypeProjectStop = 'max' and RelativeLimit is positive:

\[ \max(\| \text{div } B \|) < \text{RelativeLimit} \max(\| \text{div } B_0 \|) \]

TypeProjectStop = 'max' and RelativeLimit is negative:

\[ \max(\| \text{div } B \|) < \text{AbsoluteLimit} \]

where \( \| \cdot \| \) is the second norm, and \( B_0 \) 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 \( B_0 \), or less than the constant AbsoluteLimit.
4.2. INPUT COMMANDS FOR THE BATSRUS: GM, EE, SC, IH AND OH COMPONENTS

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

```plaintext
#PROJECTION
cg          TypeProjIter
rel 0.0001   TypeProjStop
0.0         RelativeLimit
500         MaxMatvec
```

The purpose of the correctP subroutine is to remove any discrepancies between pressure stored as the primitive variable P and the pressure calculated from the total energy E. Such discrepancies can be caused by the constrained transport scheme and by the projection scheme which modify the magnetic energy. The algorithm is the following:

Define the ratio of thermal and total energies \( q = \frac{E_{\text{thermal}}}{E} \) and

- If \( q < p_{\text{RatioLow}} \) then E is recalculated from P
- If \( p_{\text{RatioLow}} < q < p_{\text{RatioHigh}} \) then both P and E are modified depending on q
- If \( p_{\text{RatioHigh}} < q \) then P is recalculated from E

The second case is a linear interpolation between the first and third cases.

4.2.14 Coupling parameters

```plaintext
#RAYTRACE command
#RAYTRACE
T UseRaytrace (rest is read if true)
T UseAccurateTrace
0.1 DtExchangeRay [sec]
1 DnRaytrace
```

Raytracing (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 rays and for tracing rays.

By default UseRaytrace 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 ray exits the domain that belongs to the PE, its information is sent to the other PE where the ray 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 DtExchangeRay parameter. This is an optimization parameter for speed. Very small values of DtExchangeRay result in many exchanges with few rays, 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 DtExchangeRay = 0.1 seconds (default).

The DnRaytrace parameter contains the minimum number of iterations between two ray 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 ray tracing is always redone if the grid changes due to an AMR.

Default values are UseAccurateIntegral = .true. (if there is magnetic field), UseAccurateTrace = .false., DtExchangeRay = 0.1 and DnRaytrace=1.

**#RAYTRACELIMIT command**

```
#RAYTRACELIMIT
50 RayLengthMax
```

RayLengthMax provides the maximum length in planet radius for tracing a ray. This avoid tracing extremely long field lines that are not used later. The default is 100 planet radius.

**#IE command**

```
#IE
T DoTraceIE
```

DoTraceIE will activate accurate ray tracing on closed fieldlines 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 fieldlines are passed.

**#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 parameter 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.

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. 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

\[ u_{Orth}' = u_{Orth} + (u_{IonoOrth} - u_{Orth})/(\text{TauCoupleUiono}+1) \]

Therefore the larger TauCoupleUiono 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:

\[ u_{Orth}' = u_{Orth} + \min(1.0, dt/\text{TauCoupleUiono})*(u_{IonoOrth} - u_{Orth}) \]

where dt is the time step. It takes about 2*TauCoupleUiono seconds to get uOrth close to uIonoOrth.

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 $p_{IM}$ (and density $\rho_{IM}$), 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

$$p' = \frac{p \times \text{TauCoupleIm} + p_{IM}}{\text{TauCoupleIm} + 1}$$

Therefore the larger TauCoupleIm is the slower the adjustment will be. It takes approximately $2 \times \text{TauCoupleIm}$ time steps to get $p$ close to $p_{IM}$. In time accurate mode, the relaxation is based on physical time:

$$p' = p + \min(1.0, \frac{\Delta t}{\text{TauCoupleIm}}) \times (p_{IM} - p)$$

where $\Delta t$ is the time step. It takes about $2 \times \text{TauCoupleIm}$ seconds to get $p$ close to $p_{IM}$. If the time step $\Delta t$ exceeds TauCoupleIm, $p' = p_{IM}$ is set in a single step. The default value is TauCoupleIm=20.0, which corresponds to typical relaxation rate.

The DoImSatTrace logical decides 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 dLatSmoothIm degrees along the magnetic latitude direction. Default is -1.0, which means no
smoothing.

#MULTIFLUIDIM command

#MULTIFLUIDIM
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 parameters, 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
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).

#PICUNIT command

#PICUNIT
xUnitPicSi [m]
3000e3 uUnitPicSi [m/s] Speed of light for iPIC3D

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 iPIC3D model, since c=1 is defined. Using the true speed of light makes the
convergence slow in the implicit solver of iPIC3D. 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).
Define the number of PIC regions and their size and grid resolution. All distances are given in the BATSRUS distance units. When coupling with IPIC3D, the number of PIC cells in any dimension should be an integer multiple of the number of subdomains in that direction (see Config.pl -h in IPIC3D). The grid resolution of the PIC region can be different from the grid resolution of BATSRUS.

The default is to have no PIC regions at all, so this command is required for the MHD-EPIC algorithm.

### 4.2.15 Physics parameters

#### #GAMMA command

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4/3</td>
<td>Gamma for fluid 1</td>
</tr>
<tr>
<td>1.4</td>
<td>Gamma for fluid 2</td>
</tr>
<tr>
<td>5/3</td>
<td>Gamma for electrons (if UseElectronPressure)</td>
</tr>
</tbody>
</table>

The adiabatic index $\gamma = \frac{c_p}{c_v}$ (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 $P_{e-}$ variable in the equation module.

Default is 5/3 for all the gamma-s.

#### #PLASMA command

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>FluidMass [amu]</td>
</tr>
<tr>
<td>1.0</td>
<td>IonCharge [e]</td>
</tr>
<tr>
<td>0.0</td>
<td>ElectronTemperatureRatio</td>
</tr>
</tbody>
</table>

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 $n=\rho/\text{FluidMass}$. 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 hydrogen plasma IonCharge=0.1.

For multifluid/multispecies 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 \( T_e = T \times \text{ElectronTemperatureRatio} \) where \( T \) is the ion temperature. The total pressure \( p = n_k T + n_e k T_e \), so \( T = p/(n_k + n_e k \times \text{ElectronTemperatureRatio}) \). 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 \( pE = \text{ElectronTemperatureRatio} \times \sum(p_{\text{Ion}}) \). 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 \( n = \sum(Rho\text{Species}\_V/(\text{ProtonMass}\_\text{Species}\_V)) \).

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 unless "make")
ascii TypeFile (read this unless "make")
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
```
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 ("ascii", "real4" or "real8") of the table are read when NameCommand contains "load", "save", or "use".

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.

#UNIFORMSTATE command

#UNIFORMSTATE
0.125 Var1
1.0 Var2
-0.5 ... 0.0
1.0

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. Var1Left
0. Var2Left
0. ...
0.
0.
0.75
1.
0.
1.
0.125 Var1Right
0. Var2Right
0. ...
0.
0.
0.75
-1.
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, i.e. 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, i.e. the shock is in the X=0 plane.

#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 structured 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:
4.2. INPUT COMMANDS FOR THE BATSRUS: GM, EE, SC, IH AND OH COMPONENTS

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

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

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

UseBody (rest of parameters read if true)

3.0 rBody (user units)

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

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 1.0.

The rCurrents parameter defines where the currents are calculated for the GM-IE coupling. This only matters if BATSRUS is running as GM and it is coupled to IE.

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 values depend on the component: For GM rBody=3, rCurrents=4, BodyNDim = 5/ cc and BodyTDim = 25000 K. For SC/IH/OH: rBody=1, BodyNDim = 1.5e8/ cc, BodyTDim = 2.8e6 K. The rBody and rCurrents are given in normalized units (usually planetary or solar radii).

#STAR command

RadiusStar (in solar radius)

MassStar (in solar mass)

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.

### #GRAVITY command

**#GRAVITY**

- **UseGravity** (rest of parameters read if true)
- **iDirGravity** (0 - central, 1 - X, 2 - Y, 3 - Z direction)
- **GravitySi** [m/s²] (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.

### #ARTIFICIALVISCOSETY command

**#ARTIFICIALVISCOSETY**

- **UseArtificialViscosity**
- **AlphaVisco**
- **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**

- **UseViscosity**
- **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 description 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 = Nm²/(A²s). In BATSRUS, however, we use "normalized" units, so that the magnetic permeability [N/A²] disappears from the equations. So what is described here as "resistivity", is really \eta/\mu_0 which has units of [m²/s], same as (magnetic) diffusion. Since the normalized current is defined as curl B (instead of curl B/\mu_0), the electric field is \mathbf{E} = -\mathbf{u} \times \mathbf{B} + \eta \mathbf{J} in the normalized units.

If UseResistivity 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.

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.

#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 logica.

#RESISTIVEREGION command

#RESISTIVEREGION
+nagnetotail -nearbody StringResistRegion

This command is only useful if 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 description 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 (rest of parameters read only if true)
1.0 HallFactorMax
0.1 HallCmaxFactor

If UseHallResist is true the Hall resistivity is used.

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 between 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 description that can be used to make the transition smoother.

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.

#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

If UseElectronEntropy is true, solve for the electron entropy Se defined as Se = Pe**(1/GammaE). For discontinuous problems electron entropy should be conserved (if there are no source terms) if the shock is subsonic for the electrons (usually is).

Explicit electron heatconduction is not implemented for the electron entropy, but the semi-implicit heat conduction should work fine.

The default is UseElectronEntropy false, so the electron pressure equation is used.
#ANISOTROPICPRESSURE command

T UseConstantTau_I
10 TauInstabilitySi_I
100 TauGlobalSi_I
T UseConstantTau_I
10 TauInstabilitySi_I
100 TauGlobalSi_I

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 and also recommended.

If UseConstantTau is set to true, TauInstabilitySi provides the 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 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

-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

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**

**#HEATLUXLIMITER**

`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 collisional 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 $v_{th} = \sqrt{\frac{k_B T_e}{m_e}}$ is a characteristic thermal velocity, then the threshold model limits the heat conduction flux $F = -\kappa \text{grad}(T_e)$, with heat conduction coefficient $\kappa$, by $F = -\min(\kappa, f F_{fs} / \text{grad}(T_e)) \text{grad}(T_e)$ 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 quantities 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**

`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-geometry.

For TypeBeam=rz, each beam consists of $2n_{RayPerBeam}+1$ 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, resulting in a heat conduction coefficient of $9.2 \times 10^{-12} \text{ W m}^{-1} \text{ K}^{-7/2}$.

Fully ionized hydrogen plasma is assumed.

For TypeHeatConduction='user' the heat conduction coefficient of the field-aligned heat conduction is read from the user_material_properties subroutine in the ModUser module. Optional parameters should be read with subroutine user_read_inputs of the ModUser file.

The default is UseHeatConduction=.false.
#IONHEATCONDUCTION command

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, resulting in a heat conduction coefficient of

\[ 2.6 \times 10^{-13} \ W \ m^{-1} \ K^{-7/2} \]

for protons.

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

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

\[ f_S = \frac{1}{1 + (r/r_{\text{Collisional}})^2}. \]

If rCollisionless is positive, then the electron heat conduction coefficient is multiplied by

\[ f_S = \exp(-(r - r_{\text{Collisional}})/(r_{\text{Collisionless}} - r_{\text{Collisional}})) \ast 2). \]

In both cases, if the #HEATFLUXCOLLISIONLESS command is set, then the polytropic index in the electron pressure equation is smoothly interpolated between \( \gamma \) in the collisional regime and \( \gamma_H \) in the collisionless regime:

\[ \gamma_e = \gamma f_S + \gamma_H (1 - f_S), \]

where \( \gamma_H \) is defined in the #HEATFLUXCOLLISIONLESS command.

The default is UseHeatFluxRegion=.false.

#HEATFLUXCOLLISIONLESS command

If the #HEATFLUXCOLLISIONLESS command is set, then the polytropic index in the electron pressure equation is smoothly interpolated between \( \gamma \) in the collisional regime and \( \gamma_H \) in the collisionless regime:

\[ \gamma_e = \gamma f_S + \gamma_H (1 - f_S), \]

where \( \gamma_H \) is defined in the #HEATFLUXCOLLISIONLESS command.

The default is UseHeatFluxRegion=.false.
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 \( \alpha \) 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.

### #SECOND BODY command

```plaintext
#SECOND BODY

T UseBody2 ! Rest of the parameters read if .true.
0.01 rBody2
-40. xBody2
0. yBody2
0. zBody2
0.011 rCurrents2 !This is unused currently
5.0 RhoDimBody2 (/ccm) density for fixed BC for rho_BLK
25000.0 TDimBody2 (K) temperature for fixed BC for P_BLK
T UseOrbit
365.25 OrbitPeriod [days]
```

Defines the radius, (initial) position, surface density and temperature, and orbit period (if any) 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. Default is UseBody2=.false.

### #DIPOLEBODY2 command

```plaintext
#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.

### 4.2.16 Corona specific commands

### #MAGNETOGRAM command

```plaintext
#MAGNETOGRAM

T UseMagnetogram (rest of parameters read if true)
1.0 rMagnetogram
2.5 rSourceSurface
0.0 HeightInnerBc
Param/CORONA/CR1935_WSO.dat NameHarmonicsFile
```
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. nHeaderLine is the number of header lines in the harmonics file.

PhiShift is the offset in the longitude (calculated as the difference in longitude of the magnetogram central meridian and 180 degrees. PhiShift=0 means that the synoptic map is averaged over the whole Carrington rotation. With PhiShift = -1.0 the code calculates the actual value for PhiShift by extracting the central meridian coordinate from the harmonics file.

UnitB may be used both as a fudge-factor to upscale the (usually reduced) observational data and to accomodate data measured in the units different from Gauss. Wilcox data (in micro-Tesla) may be only used with UnitB=0.01-0.03.

Default is UseMagnetogram=F.

# READPOTENTIALFIELD command

#READPOTENTIALFIELD
T UseMagnetogram (rest of parameters are read if true)
potentialfield.out NamePotentialFieldFile
0.0 HeightInnerBc
3.0 UnitB

If UseMagnetogram=T then read the potential field from a file.

NamePotentialFieldFile provides the name of the file containing the 3D potential field solution, typically produced by the FDIPS code.

HeightInnerBc is the height above the photosphere of the boundary surface, non-zero values for this parameter are not recommended to unexperienced users.

UnitB may be used both as a fudge-factor to upscale the (usually reduced) observational data and to accomodate data measured in the units different from Gauss. Wilcox data (in micro-Tesla) may be only used with UnitB=0.01-0.03.

# 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 increments 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
4.2. INPUT COMMANDS FOR THE BATSRUS: GM, EE, SC, IH AND OH COMPONENTS

#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 shown.

#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 $w$ (for positive radial magnetic field $B_r$ and $w_-$ for negative $B_r$) then follows from $S_A = V_A w \propto B_\odot^2$, 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

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

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
T UseWaveReflection (read for turbulentcascade only)
1.5e5 LperpTimesSqrtBSi (read for turbulentcascade 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 solar 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 dissipation 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, \( |z_-| = 2\sqrt{w_-/\rho} \), 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. The UseWaveReflection logical is obsolete, but kept for compatibility.

The default is TypeCoronalHeating="none"

# 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 at r=1.0 [Rsun] and DecayLengthCh is the e-folding length in units of Solar Radii. The idea is to use this command 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/cm³/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

DoOpenClosedField

T

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.

#HEATPARTITIONING command

HEATPARTITIONING

uniform TypeHeatPartitioning
0.6 QionRatio
0.0 QionParRatio (if used)

stochasticheating TypeHeatPartitioning
0.17 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 Alfvén 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.
CHAPTER 4. COMPLETE LIST OF INPUT COMMANDS

See van der Holst et al. (2014) for details of the StochasticExponent and StochasticAmplitude input parameters.

#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.

#FIELDLINETHREAD command

#FIELDLINETHREAD
T UseFieldLineThreads
45 nPointThreadMax
0.002 DsThreadMin

#THREADEDDBC command

#THREADEDDBC
T UseAlignedVelocity
F DoConvergenceCheck
limited TypeBc

#TRANSITIONREGION command

#TRANSITIONREGION
T DoExpandTransitionRegion
3.0E+5 TeTransitionRegionSi
1.0E+4 DeltaTeSi (read if DoExpandTransitionRegion 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 $Te_{TransitionRegionSi}$ 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 DoExpandTransitionRegion 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 $Te=Te_{TopTransitionRegion}$.

Default value is DoExpandTransitionRegion = .false.

4.2.17 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 $\text{DoThinCurrentSheet} = \text{.false.}$

### #HELIUPDATEB0 command

**#HELIUPDATEB0**

-1.0 $\text{DtUpdateB0 [s]}$

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**

-3.0 $\text{HelioDipoleStrength [G]}$
0.0 $\text{HelioDipoleTilt [deg]}$

Variable $\text{HelioDipoleStrength}$ defines the equatorial field strength in Gauss, while $\text{HelioDipoleTilt}$ is the tilt relative to the ecliptic North (negative sign means towards the planet) in degrees.

Default value is $\text{HelioDipoleStrength} = 0.0$.

### #HELIBUFFERGRID command

**#HELIBUFFERGRID**

19.0 $\text{rBuffMin}$
21.0 $\text{rBuffMax}$
45 $\text{nThetaBuff}$
90 $\text{nPhiBuff}$

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 choosen 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.

### 4.2.18 Wave specific commands

### #ADVECTWAVES command

**#ADVECTWAVES**

T $\text{DoAdvectWaves}$

If $\text{DoAdvectWaves} = \text{.true.}$, 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.
CHAPTER 4. COMPLETE LIST OF INPUT COMMANDS

#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.

4.2.19 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 #IN-
CLUDE 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

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

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

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

IE/Plots NameIonoDir

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

#SAVEPLOT command

2 nFile
min idl StringPlot
-1 DnOutput
10.0 DtOutput [sec]
max tec StringPlot
-1 DnOutput
20.0 DtOutput [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 DnOutput and DtOutput 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/m²], Phi [kV] The ‘max’ variable set includes: X [Re], Y [Re], Z [Re], Theta [deg], Psi [deg], SigmaH [mhos], SigmaP [mhos], E-Flux [W/m²], Ave-E [eV], Jr [mA/m²], Phi [kV], Ex [mV/m], Ey [mV/m], Ez [mV/m], Jx [microA/m²], Jy [microA/m²], Jz [microA/m²], Ux [km/s], Uy [km/s], Uz [km/s], JouleHeat [mW/m²], IonNumFlux [/cm²/s], RT 1/B [1/T], RT Rho [amu/cm³], RT P [Pa], conjugate dLat [deg], conjugate dLon [deg] The ‘uam’ variable set includes: Theta [deg], Psi [deg], SigmaH [mhos], SigmaP [mhos], Jr [mA/m²], Jr(NW) [mA/m²], E-Flux [W/m²], Ave-E [eV], Phi [kV] The ‘aur’ variable set includes: Theta [deg], Psi [deg], SigmaH [mhos], SigmaP [mhos], Jr [mA/m²], Phi [kV], E-Flux [W/m²], Ave-E [eV], RT 1/B [1/T], RT Rho [amu/cm³], RT P [Pa], JouleHeat [mW/m²], IonNumFlux [/cm²/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
150.0 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 Polar-CapPedConductance, 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.

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 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.

**#GEOMAGINDICES command**

**#GEOMAGINDICES**

THIS COMMAND IS DEPRECIATED. Control of virtual magnetometers and related outputs is handled by the GM component only.

This command activates the ionospheric contribution to GM’s calculation of geomagnetic indices, activated by the GM command of the same name. While it is possible to run the GM version without an IE contribution, it is not recommended. This command currently has no parameters associated with it.

**#MAGNETOMETER command**

**#MAGNETOMETER**

magin.dat NameMagInputFile
-1 DnOutput
60.0 DtOutput

THIS COMMAND IS DEPRECIATED. Control of virtual magnetometers and related outputs is handled by the GM component only.

The #MAGNETOMETER command is used for the calculation of the ground perturbations caused by two current systems: Hall and Pedersen currents. In addition, the effects of the field aligned currents in the 'gap' region and the magnetospheric currents in the GM domain can be calculated with the equivalent command in the GM section. If the IE is coupled to GM, it sends the IE contributions to GM, so GM can save them together.

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**

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.

The DnOutput and DtOutput parameters determine the frequency of writing out the calculated perturbations in number of potential solves and time intervals, respectively.

The ground-based magnetic perturbations are written into the output file
IE/ionosphere/IE_mag_tYYMMDD_HHMMSS.mag

in which the number of solves, the station index, the location of the station in SM coordinates, the 3 components (magnetic northward, eastward, and downward) of the magnetic perturbations due to Hall and Pedersen currents are saved. The unit of the perturbation output is nT.

Default is no magnetic perturbation calculation.

#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 switched 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 parameter.

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. This is the default model when iConductanceModel is set to 4 or 5 in the #IONOSPHERE PARAM. Default values are shown; default files are automatically copied to the run directory upon creation. Any files read with this command are expected to be placed into the IE folder of the run 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 $\Sigma_1$, and occasionally $\Sigma_0$ 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/ion 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).

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

#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**

```plaintext
#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 overall conductance from both FACs and the aurora.

Default values are shown above.

**#BOUNDARY command**

```plaintext
#BOUNDARY
10.0 LatBoundary
```

Define the low latitude boundary for the potential solver. The default is set dynamically by the boundary of the closed field line region.

### 4.3.4 Scheme parameters

**#SOLVER command**

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

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

**#KRYLOV command**

```plaintext
#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 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.

#TIMESIMULATION command

```
#TIMESIMULATION
0.0     TimeSimulation
```

This command specifies the simulation time.

4.4.2 Input/output

#SAVEPLOT command

```
#SAVEPLOT
2     DtSavePlot
F     DoSaveFlux
F     DoSaveDrifts
F     DoSavePSD
F     UseSeparatePlotFiles
```

Define frequency of plots. When DoSaveFlux is false, only integrated values such as species pressure density etc is saved. When DoSaveFlux is true, then another file containing information needed to make plots of species flux in each energy bin is saved. Use SeparatePlotFiles defines if each time is saved in the same file or if a separate file is used for each time.

#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 of circle.
#MINIMUMPRESSURETOGM command

#MINIMUMPRESSURETOGM
1e-2 MinimumPressureToGM

Sets minimum pressure passed to GM.

#RESTART command

#RESTART
F IsRestart

Determine whether or not to continue a previous run.

#INITIALF2 command

#INITIALF2
F IsEmptyF2
T IsGmInitial
F IsDataInitial

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 (IsEmptyF2=T, not recommended) or to set the initial fluxes in the simulation to values from AMPTE/CCE data (IsDataInitial=T). The default is shown.

#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.

4.4.3 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.

#DECAY command

#DECAY
T UseDecay
36000. 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.

#COMPOSITION command

#COMPOSITION
FIXED NameCompModel
0.85 FractionH
0.15 FractionO
1.0 FractionElectron

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.
4.5 Input Commands for the CRCM: IM Component

List of IM commands used in the PARAM.in file

4.5.1 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.

#TIMESIMULATION command

#TIMESIMULATION
0.0 TimeSimulation

This command specifies the simulation time.

4.5.2 Input/output

#SAVEPLOT command

#SAVEPLOT
2 DtSavePlot
F DoSaveFlux
F UseSeparatePlotFiles

Define frequency of plots. When DoSaveFlux is false, only integrated values such as species pressure density etc is saved. When DoSaveFlux is true, then another file containing information needed to make plots of species flux in each energy bin is saved. Use SeparatePlotFiles defines if each time is saved in the same file or if a separate file is used for each time.

#SAVELOG command

#SAVELOG
10 DtSaveLog

When this command is set, a log file for CRCM 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 of circle.
#MINIMUMPRESSURETOGM command

#MINIMUMPRESSURETOGM
1e-2 MinimumPressureToGM

Sets minimum pressure passed to GM.

#RESTART command

#RESTART
F IsRestart

Determine whether or not to continue a previous run.

#INITIALF2 command

#INITIALF2
F IsEmptyF2
T IsGmInitial
F IsDataInitial

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 (IsEmptyF2=T, not recommended) or to set the initial fluxes in the simulation to values from AMPTE/CCE data (IsDataInitial=T). The default is shown.

#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.
4.6 Input Commands for the Hot Electron Ion Drift Integrator: IM Component

4.6.1 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
4.6. INPUT COMMANDS FOR THE HOT ELECTRON ION DRIFT INTEGRATOR: IM COMPONENT

#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

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

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 scaling factor for Maxwellian initial distribution. CharacteristicEnergy is the characteristic energy for Initial*1 (keV)

#OUTPUT command

DoSaveDistributionFunctionEverywhere
DoSaveEquatorialDistributionFunction
DoSaveEnergyDeposition
DoSaveTotalPrecipitationFlux
DoSaveDifferentialPrecipitationFlux
DoSaveParticleEnergyLosses
DoSaveThermalPlasmaDensity
DoSaveCflForAdvection
DoSaveDriftVelocities
DoSaveEvsLDistributions
DoSaveParticleLifetimes
4.6. INPUT COMMANDS FOR THE HOT ELECTRON ION DRIFT INTEGRATOR: IM COMPONENT

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 differential.

#OUTPUTINFO command

#OUTPUTINFO
99266 NameRun
1800. nFrequency [sec]

The #OUTPUTFREQUENCY indicates the time interval between printouts. NameRun sets the prefix of the output files. nFrequency 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
MBI-VS - MBI driven V-S, activity from force balance at dusk
**CHAPTER 4. COMPLETE LIST OF INPUT COMMANDS**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>McIlwain</td>
<td>McIlwain field</td>
</tr>
<tr>
<td>McIlwainPlusCPCP</td>
<td>McIlwain field, Eo driven by Cross Polar Cap Potential</td>
</tr>
<tr>
<td>KpVSPPlusBurkeWygant</td>
<td>Kp V-S field plus Burke-Wygant penetration field</td>
</tr>
<tr>
<td>McIlwainPlusBurkeWygant</td>
<td>McIlwain field plus Burke-Wygant penetration field</td>
</tr>
<tr>
<td>McIlwainCPCPBurkeWygant</td>
<td>McIlwain(CPCP) field plus B-W penetration field</td>
</tr>
<tr>
<td>KpVSSSelfConsistent</td>
<td>Kp V-S field plus self-consistent penetration field</td>
</tr>
<tr>
<td>McIlwainSelfConsistent</td>
<td>McIlwain field plus self-consistent penetration field</td>
</tr>
<tr>
<td>McIlwainCPCPSelfConsistent</td>
<td>McIlwain(CPCP) field plus S-C penetration field</td>
</tr>
<tr>
<td>UnshieldedVSWithPen</td>
<td>Unshielded V-S(CPCP) field plus penetration field</td>
</tr>
<tr>
<td>UnshieldedVSNoPen</td>
<td>Unshielded VS(CPCP) field (no Epen)</td>
</tr>
<tr>
<td>W96SC</td>
<td>W96 field plus S-C penetration field</td>
</tr>
<tr>
<td>W96</td>
<td>W96 field everywhere (done in S-C field block)</td>
</tr>
<tr>
<td>AMIESC</td>
<td>AMIE field plus S-C penetration field</td>
</tr>
<tr>
<td>AMIE</td>
<td>AMIE field everywhere (done in S-C field block)</td>
</tr>
<tr>
<td>ReadFromFile</td>
<td>Read in electric field from a file</td>
</tr>
<tr>
<td>AMIEPot</td>
<td>AMIE potentials everywhere</td>
</tr>
<tr>
<td>W96Pot</td>
<td>Wiemer-96 potentials everywhere</td>
</tr>
<tr>
<td>Foster</td>
<td>Foster potentials everywhere</td>
</tr>
</tbody>
</table>

### #INITIALTHERMALPLASMA command

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

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

### #PITCHANGLE command

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

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

### #BFIELD command

Either analytic or uniform is used.
4.6. INPUT COMMANDS FOR THE HOT ELECTRON ION DRIFT INTEGRATOR: IM COMPONENT

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 points are evenly distributed along the field line. For non-uniform grid, the grid is more refined in the equatorial region.

#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'.
4.7 Input Commands for the Rice Convection Model 2: IM Component

4.7.1 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.

4.7.2 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 Year-MoDy-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 format 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 ‘eta 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 ‘budloc Vm —B— V birk alam eta 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).

4.7.3 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.

4.7.4 Physics parameters

#COMPOSITION command

#COMPOSITION
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 LossFactor
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 $R_{ele} \times \text{LossFactor}$. If LossFactor=0, then there is no loss through precipitation; 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

$$\exp\left(-\frac{\delta T}{\text{DecayTimescale}}\right)$$

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.
4.8 Input Commands for the IPIC3D2

#SAVEIDL command

```plaintext
#SAVEIDL
4
sat_sat01.sat var ascii si StringPlot
   -1 DnOutput
   3.0 DtOutput
   0. DxOutput
rhoS0 rhoS1 rho pxx pxx0 pxx1 Ex Ey Ez Bx By Bz
sat_sat02.sat particles1 real4 si StringPlot
   1 DnOutput
   -0.05 DtOutput
   10. DxOutput: output one particle of every DxOutput particles.
z=0 var ascii si StringPlot
   -1 DnOutput
   3.0 DtOutput
   0. DxOutput
rhoS0 rhoS1 rho pxx pxx0 pxx1 Ex Ey Ez Bx By Bz
x=0 var ascii pic StringPlot
   1 DnOutput
   -0.05 DtOutput
   0. DxOutput
rhoS0 rhoS1 bx by pxx NamePlotVar
3d all real4 planet StringPlot
   1 DnOutput
   -0.05 DtOutput
   5 DxOutput
cut all real8 si StringPlot
   1 DnOutput
   -0.05 DtOutput
   0 xMin
   1 xMax
   2 yMin
   3 yMax
   4 zMin
   5 zMax
   0. DxOutput
cut particles0 real4 si StringPlot
   1 DnOutput
   -0.05 DtOutput
   0 xMin
   1 xMax
   2 yMin
   3 yMax
   4 zMin
   5 zMax
   1. DxOutput: output one particle of every DxOutput particles.
3d particles1 real4 si StringPlot
   1 DnOutput
```

This command determines the IDL type output from iPIC3D.

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 particles and '3d' type field output. Only one of every DnOutput nodes/particles are saved in order to save disk space, also to speed out the post processing and virtualization. Several other parameters may follow based on the content of StringPlot.

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 the coordinate.
- **z=z0** - a 2D cut of z=z0 plane. 'z0' is the coordinate.
- **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 specify the output range.
- **sat_SatelliteFile** - save information along the satellite trajectory read from SatelliteFile.

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 kXXS0 kYYS0 kZZS0 kXYs0 kYZS0 kXXS1 kYYS1 kXYZ1 kYZZ1 kXS1 kYYS1 kYZZ1 jxS0
- **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 pYXS0 pZZS0 pXYS0 pXZS0 pYZS0
- **var** - read from parameter file
- **particles0/particles1/particles2...** - the location, velocity and weight of particles belonging to species0/particles1/particles2...

The particle species needs to be specified. One and only one species is allowed in one output 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 has exclude 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 EMfields3D.cpp::getVar(). 2) DxOutput is only functional for particles and 3d field output now. 3) The position for "cut", "x=", "y="... is in BATSRSU coordinate. 4) Output variable 'particles' only works for 'cut', '3d' and 'sat'. 5) The unit of current and electric field are not well defined so far.

##SATELLITERADIUS command

###SATELLITERADIUS

2 SatRadius

If the distance between a particle and the satellite trajectory is less than SatRadius*dx, where dx is the cell size in x direction, this particle will be saved in the output file.
#SMOOTH command

#SMOOTH
T    DoSmoothAll
4    nSmooth
0.5  InnerSmoothFactor
1.0  BoundarySmoothFactor
5    nBoundarySmooth

These parameters determine how to smooth electric field and optionally the cell centered total current and total charge density (hat functions).

If DoSmoothAll is true, all three quantities are smoothed. If DoSmoothAll is false, only the electric field will be smoothed.

The algorithm for smoothing is:

\[ E'_{ijk} = \text{smooth} \times (E_{ijk}) + \frac{(1 - \text{smooth})}{6} \times \text{sum(six neighbors of } E_{ijk}) \]

This process will repeat nSmooth times. The coefficient ‘smooth’ is calculated based on innerSmoothFactor, boundarySmoothFactor and nBoundarySmooth and the distance of the cell from the boundary. Define iCell as the distance from a node to the PIC boundary (node index 1 NOT 0). The coefficient ‘smooth’ will be linearly interpolated from innerSmoothFactor and boundarySmoothFactor.

The value of ‘smooth’ calculated from innerSmoothFactor and boundarySmoothFactor is limited to be smaller than 1 and larger than 0.

Default is nSmooth=0, so no smoothing is done.

#SUBCYCLING command

#SUBCYCLING
T    doSubCycling

If doSubCycling is true, the time step for the update of each particle will field strength.

#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.

#EMWAVE command

#EMWAVE
T    doTestEMWave
1.0  Lx
1e99  Ly
2.0  Lz
30   Phase
2.0  AmplEx
1.0  AmplEy
0    AmplEz
If doTestEMWave is true, a EM field will be initialized as a pure EM wave, and the feedback from particles will be ignored. The E field lengths Lx, Ly and Lz are in SI unit. The wave vector (for example, $2\pi/Lx$) will be set to 0 if the corresponding wave length is larger than $1e10$. The unit of the initial phase 'Phase' is 'degree'. The E fields are initialized as:

$$E = (AmplEx, AmplEy, AmplEz) \cdot \sin(kx + \text{Phase} \cdot \text{deg2rad})$$

The magnetic field is set from $B = k/\pi - \mathbf{k} \cdot \mathbf{E}$.

### #SOLVER command

```
#SOLVER
1e-8 GMREStol
100 nGMRESRestart
3 NiterMover
```

Warning: this command is obsolete because the linear solver in SWMF is used by default, and the related parameters are set by command #EFIELDSSOLVER and #POISSON.

These parameters control the field solver and the particle mover. 'GMREStol' is the tolerance of the GMRES E field solver, which will restart every 'nGMRESRestart' cycles. The maximum iteration number of the implicit particle mover is 'NiterMover'.

Note: 1) The tolerance for the CG iteration used to be included in this command. It is removed because CG is not used anymore. 2) The tolerance of the Poisson solver is hard coded so far. It's default value is 0.1.

### #EFIELDSSOLVER command

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

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

### #POISSON command

```
#POISSON
1 DoCorrection
1 PoissonCorrectionCycle
0.1 PoissonTol
20 PoissonIter
```

This command is off by default.

### #ENERGYCONSERVING command

```
#ENERGYCONSERVING
T useECSIM
```

If useECSIM is true, then the Energy Conserving Semi-Implicit Method (ECSIM) (Lapenta, JCP, 2016) will be used.
#DISCRETIZATION command

#DISCRETIZATION

0.5 theta
0.0 gradRhoRatio
0.0 cDiff

This command should be used after #ENERGYCONSERVING. For ECSIM, theta=0.5 is the default to achieve the exact energy conserving. In practice, theta=0.51 has good energy conserving property and can also help to supress numerical noise and instabilities. For iPIC3D, theta=1 is the default.

‘gradRhoRatio’ is the coefficient of a term used to supress the (div(E)-charge) error. It is between 0 and 1. ‘cDiff’ is a positive coefficient of the numerical diffusion that is used to smooth the electric field. cDiff=1 is a reasonable choice.

See iPIC3D-ECSIM Algorithm.tex for more details.

#DIVE command

#DIVE

weight_estimate

#DIVE

weight divECleanType
1 nPower
1e-8 divECleanTol
50 divECleanIter

#DIVE

position_light divECleanType
1 nPower
1e-8 divECleanTol
50 divECleanIter
3 nIterNonLinear

#DIVE

position_all divECleanType
1 nPower
1e-8 divECleanTol
50 divECleanIter
3 nIterNonLinear

#DIVE

position_estimate_phi divECleanType

Changing the particle postions by adding a small displacement dx, or the weights by multiplying the weight by a vaule rp close to 1, to satisfy the Gaussi’s law: div(E)=net charge. The correction can be done by estimating the changes (weight_estimate, position_estimate,phi) or calculating the changes accurately (weight, position_light, position_all). The accurate corrections need to solve an linear equation system, so the tolerance and the maximum iteration number should be specified. For the position corrections, the equation system is non-linear, and the non-linear iteration number nIterNonLinear should be set.

For the accurate corrections, \( \sum(dx^2 \cdot qp^n Power) \) or \( \sum((rp - 1)^2 \cdot qp^n Power) \) are minimized for position or weight correction, respectively. nPower=1 is the optimized choice.

‘position_light’ or ‘weight’ only correct the lightest species, which is usually the electron, and ‘position_all’ corrects all species.
#CELLCENTERDENSITY command

F doCalcRhocDirectly

Calculating cell center densities by particle-cell center interpolation, or interpolations from node densities. Default is false, but it is turn on if #DIVE is used.

#TIMESTEPPING command

F useSWMFDt

The three possible ways to use this command is listed above. If useSWMFDt is true, the PIC code will use the the timestep obtained from SWMF, which is the coupling period. When useSWMFDt is false, the third command sets a fixed time step or a CFL number if useFixedDt is true or false, respectively.

#ELECTRON command

-100 qom

This command sets the charge per mass in normalized unit for electrons. It will be ignored when PIC is coupled with five-moment or six-moment MHD.

#CHECKSTOP command

2 maxUth

If the maximum thermal velocity of any species exceeds maxUth, which is in normalized PIC unit and has default value 2, then the code will save the output data and stop running.
4.9 Input Commands for the DGCPM: PS Component

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

4.9.1 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.

4.9.2 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.

4.9.3 Physical parameters

#KP command

#KP const NameSourceKp 3 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
2.0E12 FluxMax

Set parameters that control flux tube refilling rate.

4.9.4 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.
4.10 Input Commands for the PWOM: PW Component

List of PW commands used in the PARAM.in file

4.10.1 Numerical scheme

#SCHEME command

```plaintext
#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. Type Flux 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

```plaintext
#LIMITER
1.5 LimiterBeta
```

Sets the beta parameter in the Rusanov scheme.

#VERTICALGRID command

```plaintext
#VERTICALGRID
390 nPoints
2e6 DeltaR
```

Sets the number of grid points in the vertical direction and the grid spacing.

#STARTTIME command

```plaintext
#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.
4.10. INPUT COMMANDS FOR THE PWOM: PW COMPONENT

#TIMEACCURATE command
#TIMEACCURATE
T DoTimeAccurate

DoTimeAccurate determines if the field lines are solved in time accurate mode or in steady state mode.

#TIMESTEP command
#TIMESTEP
50 DtHorizontal

DtHorizontal is the timestep for horizontal motion of the field line. Default value is shown.

#VARIABLEDT command
#VARIABLEDT
F IsVariableDt

When IsVariableDt=T then the vertical timestep 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.

#FAC command
#FAC
F UseJr

UseJr determines whether to use field aligned currents to affect the ion outflow. The default is shown.

4.10.2 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.
CHAPTER 4. COMPLETE LIST OF INPUT COMMANDS

#FIELDLINE command

#FIELDLINE
1 nTotalLine

nTotalLine sets the number of field lines included in the simulation. The default is shown.

#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.

#SAVEPLOT command

#SAVEPLOT
10.0 DtSavePlot
-1 DnSavePlot
T DoSaveFirst

The frequency which plot files are written out are defined here. As is whether or not to save a plot on the first call. The default values are given.

#SAVEPLOTELECTRODYNAMICS command

#SAVEPLOTELECTRODYNAMICS
F DoPlotElectrodynamics
10 DtPlotElectrodynamics

DoPlotElectrodynamics determines whether the electrodynamics plot information is saved. For instance the field aligned currents, and the polar cap potential can be saved with this command. The frequency of output is determined by the DtPlotElectrodynamics parameter.

#STATICATMOSPHERE command

#STATICATMOSPHERE
T UseStaticAtmosphere

To keep the MSIS atmosphere constant set UseStaticAtmosphere to True.

#MSISPARAM command

#MSISPARAM
60 F107
60 F107A
4 AP(1)
4 AP(2)
4 AP(3)
4 AP(4)
4 AP(5)
4 AP(6)
4 AP(7)

This command sets the MSIS parameters for the neutral atmosphere. The defaults are shown.
#NGDC_INDICES command

ApFile.dat NameNgdcFile

Read the Ap Index from a file and calculate the appropriate array of AP indices to feed to MSIS.

#NOAAHPI_INDICES command

HpiFile.dat NameHpiFile

Read the HPI from a file. Used to get the Auroral ionization and heating

#HPI command

HemisphericPower

Set the HPI to a constant value. Used to get the Auroral ionization and heating

#AURORA command

UseAurora

Set whether or not to include auroral ionization and bulk heating.

#SOLARWIND command

bx by bz 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

UpstreamFile

Read the IMF from a file. Unlike the SOLARWIND command, the imf can be time dependant in this case.

#END command

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.
#STOP command

```plaintext
#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.

#SE command

```plaintext
#SE
T      DoCoupleSE
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.
4.11 Input Commands for the RBE: RB Component

List of RB commands used in the PARAM.in file

4.11.1 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.

4.11.2 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  IsRestart

Determine whether or not to continue a previous run.
#END command

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.
4.12 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.

#COMPDOMAINS command

Sets the ranges of heliocentric distances for domains in SC and IH exchanging data with SP.

#COMPDOMAINS

1.15 RScMin [Rs]  
19.0 RIhMin [Rs]  
21.0 RScMax [Rs]  
220.0 RIhMax [Rs]

#ORIGIN command

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.

#ORIGIN

2.5 ROrigin [Rs]  
0.0 LonMin [deg]  
-70.0 LatMin [deg]  
360.0 LonMax [deg]  
70.0 LatMax [deg]

4.12.1 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.

#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 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.

### 4.12.2 Stopping criteria

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

### #STOP command

#STOP

- nIterMax
- 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

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.

### #CPUTIMEMAX command

#CPUTIMEMAX

CpuTimeMax [sec]

This command is only used in stand alone mode.
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.

### 4.12.3 Grid

Parameters of the Lagrangian grid used in MFLAMPA

#### #CHECKGRIDSIZE command

Check if the configured parameters of the Lagrangian grid match those assumed in the input files

```
#CHECKGRIDSIZE
1000 nParticleMax
   4 nLon
   4 nLat
```

#### #COORDSYSTEM command

Type of heliocentric coordinate system used in MFLAMPA (HGI, HGR, HGC).

```
#COORDSYSTEM
HGI TypeCoordSystem
```

### 4.12.4 Advance

#### #INJECTION command

```
#INJECTION
5.0 SpectralIndex
1.0 Efficency
```

#### #CFL command

```
#CFL
1.0 Cfl
```

### 4.12.5 Action

#### #RESTART command

```
#RESTART
T DoRestart
```

The default values are shown.

#### #DORUN command

```
#DORUN
T DoRun
```

The default values are shown.
4.12.6 Input/output

#USEDATETIME command

#USEDATETIME
T UseDateTime

Set the format for the output file names like eYYYYMMDDHHMMSS tagging the file with real date and UT time 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).

#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, because the same output files and tags were saved at the end of previous run.

#MHDATA command

#DOMHDATA
T DoReadMhData

The default values are shown.
4.13 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.

4.13.1 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**

```plaintext
#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**

```plaintext
#TIMEEND
1999 iYear
03 iMonth
25 iDay
00 iHour
00 iMinute
00 iSecond
```

This command is only used in the stand alone mode.

The #TIMEEND command sets the final date and time for the simulation.

**#RESTART command**

```plaintext
#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**

```plaintext
#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.

4.13.2 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 all of the species densities are set using MSIS. There are 2 species which MSIS does not include: [NO] and [N(^2D)]. 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).

If 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.
4.13.3 Indices

#F107 command

#F107
150.0  f107
150.0  f107a

The $f_{10.7}$ 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.

4.13.4 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.
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.

The first method only inputs the solar wind velocity, density, temperature and IMF.

The second method takes data from the NOAA SPIDR interface. You can download almost all of the parameters in this format.

The third method only accepts HPI data from the NOAA satellites.

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).
4.13.6 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 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.

4.13.7 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 Stretch-
ingPercentage: 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
4.13.8 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

CON
  #CHECKKILL, 46
  #CHECKSTOP, 47
  #CHECKSTOPFILE, 47
  #COMPONENT, 50
  #COUPLE1, 51
  #COUPLE1SHIFT, 51
  #COUPLE1TIGHT, 52
  #COUPLE2, 51
  #COUPLE2SHIFT, 52
  #COUPLE2TIGHT, 52
  #COUPLEORDER, 50
  #CYCLE, 50
  #DESCRIPTION, 44
  #DIPOLE, 57
  #ECHO, 54
  #END, 44
  #ENDTIME, 45
  #FLUSH, 54
  #IDEALAXES, 57
  #INCLUDE, 44
  #MAGNETICAXIS, 57
  #MAGNETICCENTER, 57
  #NSTEP, 46
  #PLANET, 56
  #PRECISION, 49
  #PROGRESS, 49
  #RESTARTFILE, 54
  #RESTARTOUTDIR, 53
  #ROTATEHGI, 55
  #ROTATEHGR, 55
  #ROTATION, 56
  #ROTATIONAXIS, 56
  #RUN, 45
  #SAVERESTART, 53
  #STARTTIME, 45
  #STDOUT, 54
  #STDOUTDIR, 55
  #STOP, 46
  #STRIC, 44

  #TEST, 48
  #TESTPROC, 48
  #TIMEACCURATE, 45
  #TIMESIMULATION, 46
  #TIMESTEP, 58
  #TIMING, 48
  #UPDATEB0, 57
  #VERBOSE, 48
  #VERSION, 49

EE, GM, SC, IH, OH / BATSRUS
  #ACTIVEREGIONHEATING, 157
  #ADAPTIVELOWORDER, 121
  #ADVECTWAVES, 159
  #ALFVENWAVES, 160
  #AMR, 116
  #AMRCRITERIA, 117
  #AMRCRITERIALEVEL, 118
  #AMRINITPHYSICS, 110
  #AMRLIMIT, 115
  #AMRRESOLUTION, 115
  #ANISOPRESSUREIM, 138
  #ANISOTROPICPRESSURE, 149
  #ARTIFICIALVISCOSITY, 145
  #BEGIN_COMP, 60
  #BIERMANNBATTERY, 147
  #BLOCKLEVELSRELOADED, 68
  #BODY, 144
  #BORIS, 132
  #BORISREGION, 132
  #BORISSIMPLE, 132
  #BOUNDARYSTATE, 72
  #BOXBOUNDARY, 71
  #BUFFERGRID, 75
  #CHANGEVARIABLES, 67
  #CHECKGRIDSIZE, 67
  #CHECKSTOPFILE, 92
  #CHECKTIMESTEP, 125
  #CLIMIT, 131
  #COARSEAXIS, 82
  #CODEVERSION, 66
INDEX

#COLLISION, 129
#COMPONENT, 59
#CONSERVATIVECRITERIA, 124
#CONSERVEFLUX, 123
#CONTROLDECREASE, 126
#CONTROLFACTOR, 127
#CONTROLINCINCREASE, 126
#CONTROLINIT, 126
#CONTROLTIMESTEP, 125
#CONTROLVAR, 126
#COORDSYSTEM, 79
#CORONALHEATING, 155
#CORRECT, 135
#CPCPBOUNDARY, 77
#CPUTIMEMAX, 92
#DEBUG, 66
#DESCRIPTION, 59
#DIPOLE, 63
#DIPOLEBODY2, 153
#DIVB, 133
#DIVBSOURCE, 134
#DOAMR, 115
#ECHO, 59
#ELECTRONENTROPY, 148
#ELECTRONPRESSURE, 148
#EMPIRICALSW, 155
#END, 61
#ENDTIME, 92
#END_COMP, 61
#EQUATION, 66
#EXTRABOUNDARY, 76
#EXTRAENERGY, 149
#FACEBOUNDARY, 76
#FIELDLINETHREAD, 158
#FIXAXIS, 81
#FIXEDTIMESTEP, 84
#FLATTENING, 122
#FLUSH, 110
#GAMMA, 139
#GEOMAGINDICES, 98
#GRAVITY, 145
#GRID, 78
#GRIDBLOCK, 78
#GRIDGEOMETRY, 80
#GRIDGEOMETRYLIMIT, 80
#GRIDRESOLUTION, 113
#GRIDSYMMETRY, 79
#HALLREGION, 147
#HALLRESISTIVITY, 147
#HEATCONDUCTION, 151
#HEATFLUXCOLLISIONLESS, 152
#HEATFLUXLIMITER, 150
#HEATFLUXREGION, 152
#HEATPARTITIONING, 157
#HELIODIODPOLE, 159
#HELIODPOLE, 159
#HELIOUPDATEB0, 159
#HYPERBOLICDIVB, 134
#HYPERBOLICDIVE, 133
#IDEALAXES, 63
#IE, 136
#IMCOUPLING, 136
#IM, 137
#IMCOUPLINGSMOOTH, 138
#IMPLCHECK, 88
#IMPLICIT, 86
#IMPLICITCRITERIA, 87
#IMPLICITENERGY, 87
#IMPLSCHEME, 88
#IMPLSTEP, 88
#INCLUDE, 160
#INNERBCPE2P, 74
#INNERBOUNDARY, 73
#IONHEATCONDUCTION, 152
#IUNITS, 69
#JACOBIAN, 89
#KRYLOV, 90
#KRYLOVSIZE, 91
#LASERBEAM, 151
#LASERBEAMPROFILE, 151
#LASERBEAMS, 150
#LASERPULSE, 150
#LDEM, 154
#LIGHTSPEED, 133
#LIMITER, 131
#LIMITRADIUS, 81
#LONGSCALEHEATING, 156
#LOOKUPTABLE, 140
#LOWORDERREGION, 121
#MAGNETICAXIS, 62
#MAGNETICCENTER, 62
#MAGNETICINNERBOUNDARY, 75
#MAGNETOGRAM, 153
#MAGNETOMETER, 98
#MAGNETOMETERGRID, 99
#MIDLOADING, 151
#MESSAGEPASS, 129
#MHDIONS, 128
#MINIMUMDENSITY, 147
#MINIMUMPRESSURE, 148
#MINIMUMTEMPERATURE, 148
# TIMESTEPPING, 83
# TIMING, 68
# TRANSITIONREGION, 158
# TVDRESCHANGE, 130
# UNIFORMAXIS, 81
# UNIFORMSTATE, 141
# UPDATEB0, 63
# UPDATECHECK, 125
# USEB0, 133
# USECURLB0, 133
# USERFLAGS, 63
# USERINPUTBEGIN, 64
# USERINPUTEND, 64
# USERMODULE, 66
# VERBOSE, 66
# VISCOITY, 145
# VISCOITYREGION, 145
# WSACOEFF, 155
# YOUNGBOUNDARY, 77

IE/Ridley_serial

# AMIEFILES, 164
# AURORALOVAL, 165
# BACKGROUND, 165
# BOUNDARY, 166
# CONDUCTANCE, 166
# CONDUCTANCEFILES, 165
# DEBUG, 161
# GEOMAGINDICES, 163
# IM, 164
# IONODIR, 161
# IONOSPHERE, 162
# KRYLOV, 166
# MAGNETOMETER, 163
# RESTART, 161
# SAVELOGFILE, 162
# SAVELOGNAME, 162
# SAVEPLOT, 161
# SAVEPLOTNAME, 162
# SOLVER, 166
# SPS, 164
# STRICT, 161
# UA, 164

IM/CIMI

# COMPOSITION, 169
# DECAY, 168
# END, 168
# INITIALF2, 168
# LIMITER, 167
# MINIMUMPRESSURETOGM, 168
# RESTART, 168
# SAVELOG, 167
# SAVEPLOT, 167
# STRONGDIFFUSION, 168
# TIMESIMULATION, 167
# TYPEBOUNDARY, 167

IM/CRCM

# END, 171
# INITIALF2, 171
# LIMITER, 170
# MINIMUMPRESSURETOGM, 171
# RESTART, 171
# SAVELOG, 170
# SAVEPLOT, 170
# TIMESIMULATION, 170
# TYPEBOUNDARY, 170

IM/HEIDI

# BFIELD, 176
# BOUNDARY, 174
# CONVECTION, 175
# ENERGYSETUP, 173
# GRID, 172
# INCLUDEWAVES, 176
# INDICES, 173
# INITIAL, 174
# INITIALTHERMALPLASMA, 176
# INJECTIONFREQUENCY, 175
# INNERBOUNDARY, 173
# OUTPUT, 174
# OUTPUTINFO, 175
# PITCHANGLE, 176
# SAVERESTART, 177
# SOLARWIND, 176
# SPECIES, 173
# STARTTIME, 172
# STOP, 172
# STORM, 173
# TIMESTEP, 172

IM/RCM2

# ASCII, 178
# CHARGEEXCHANGE, 180
# COMPOSITION, 179
# DECAY, 181
# IONOSPHERE, 180
# OUTERBOUNDARY, 180
# PRECIPITATION, 180
# RCMDIR, 178
# RESTART, 179
# SAVEPLOT, 178
# SAVEPLOTNAME, 178
# STRICT, 178
# TEMPERATURE, 181
INDEX

#Timestep, 179

PC/IPIC3D2
#CellCenterDensity, 187
#CheckStop, 187
#Discretization, 186
#Dive, 186
#EFIELDSolver, 185
#Electron, 187
#EMWave, 184
#EnergyConserving, 185
#Periodicity, 184
#Poisson, 185
#SatelliteRadius, 183
#SaveIDL, 182
#Smooth, 184
#Solver, 185
#Subcycling, 184
#Timestepping, 187

PS/DGPCM
#Filling, 188
#KP, 188
#MLTSLICE, 188
#Timestep, 188

PW/PWOM
#Aurora, 193
#End, 193
#Fac, 191
#Fieldline, 192
#HPI, 193
#Limiter, 190
#MHDIndices, 193
#Motion, 191
#MSISparam, 192
#NGDCIndices, 193
#NOAAHPLIndices, 193
#Restart, 192
#Rotation, 191
#Saveplot, 192
#SaveplotElectrodynamics, 192
#Scheme, 190
#SE, 194
#SolarWind, 193
#Starttime, 190
#StaticAtmosphere, 192
#Stop, 194
#Test, 191
#TimeAccurate, 191
#Timestep, 191
#VariableEdt, 191
#VerticalGrid, 190

RB/RBE
#Bmodel, 195
#End, 197
#IEModel, 196
#InputData, 196
#Limiter, 195
#Plasmasphere, 196
#PlotElectrodynamics, 196
#Restart, 196
#Saveplot, 196
#Species, 195
#Splitting, 195
#StartupTime, 196
#Timesimulation, 195
#Timestep, 195

SP/MFLAMPA
#Begin, 198
#CFL, 200
#CheckGridSize, 200
#CheckStopFile, 199
#CompDomains, 198
#CoordSystem, 200
#CputimeMax, 199
#DORun, 200
#End, 199
#End_Comp, 198
#Injection, 200
#MhdData, 201
#Origin, 198
#Restart, 200
#Run, 199
#SaveInitial, 201
#Starttime, 198
#Stop, 199
#Usedatetime, 201

UA/GITM2
#Altitude, 207
#AmieFiles, 205
#Apex, 203
#Chemistry, 207
#CputimeMax, 203
#Debug, 205
#Diffusion, 206
#Electrodynamics, 207
#F107, 204
#Forcing, 206
#Grid, 207
#HPI, 204
#Initial, 203
#IONFORCING, 206
#ISTEP, 202
#KP, 204
#MHD_INDICES, 205
#NGDC_INDICES, 205
#NOAAHPI_INDICES, 205
#RESTART, 202
#SATELLITES, 208
#SAVEPLOT, 208
#SOLARWIND, 204
#STRETCH, 207
#THERMO, 206
#TIMEEND, 202
#TIMESTART, 202
#TSIMULATION, 203