STANDARDS

David Chesney and Gábor Tóth

May 5, 2024

Contents

Chapter 1

Data Naming Standard

1.1 Introduction

The Center for Space Environment Modeling (CSEM) is developing a software framework which consists of science modules representing various physical domains, and the framework itself, which connects the science modules. It is important to develop some data naming standards so that the independently developed science modules and the framework can be compiled together. Also it is useful to have consistent naming when many developers work together on the same part of the software.

The naming standard refers to all variable, procedure and file names used in the framework and the science modules. Here *variable name* means any constant, variable, type, structure, object, class or Fortran 90 module name, while *procedure name* means subroutine, function, or method names in general.

The purposes of the naming standard are the following:

- Avoid name conflicts between various modules of the framework.
- Extensibility for case when intra-module dataflow name is modified to inter-module dataflow.
- Characteristics of variables, procedures and files are explicitly understood from the name.
- Allow consistent names within modules developed my more than one programmer.
- Improved code readability allows outside programmers to be used.
- Language (F90, C++, etc.) independence of variable and procedure names.
- Representation in Backus-Naur Form (BNF), and thus, ability to apply syntax checking tools.

1.2 Basic Considerations

The science modules are independent in terms of variables, on the other hand, in the current bottom-up design the procedures of all modules will be compiled into a single executable. This means that modules must avoid procedure name conflicts.

For example the B0 module will be present on all processors, and it has to have procedure names that differ from the procedure names of the magnetosphere, ionosphere, heliosphere etc. modules. This restriction does not exclude the possibility of using more than one instance of the same general module. For example, if both the magnetosphere and the heliosphere are modeled by the same general magnetohydrodynamic module, the procedure names will be the same, thus these module *must* run on separate sets of processors.

Since the science modules are mostly developed by scientists and not by software engineers, it seems reasonable to minimize the constraints on the data naming. The minimum requirements seem to be the following

- All procedure names of a module must contain a unique identifier for the module.
- The control module should be able to define or modify all file names (e.g. input parameter and output data file names) used by the other modules.

The procedure names must start with the two-letter module name acronyms, and the control module will use a directory name containing the same acronym for the input and output files for each module.

An important simplification is that we allow for *intra- and inter-module* naming standard. When a single science module is being developed, one may use the intra-module naming system. When the science modules are coupled together, an automatic Perl script can add the unique prefixes for each procedures in each modules.

1.3 Physics and Software Modules

Physics modules represent a certain physics domain. There are only a well defined set of these modules, which all have two character abbreviations:

CE Cometary Environment

EE Eruptive Events

GM Global Magnetosphere

IE Ionosphere Electrodynamics

IH Inner Heliosphere

IM Inner Magnetosphere

IN Interstellar Neutrals

IO Ionosphere

OH Outer Heliosphere

PL Plasmasphere

PS Planetary Satellites

PW Polar Wind

RB Radiation Belt

SC Solar Corona

TH Thermosphere

We also define `MH` for the generic magnetohydrodynamic physics module, which can model GM, IH, SC, OH, and possibly EE.

In addition to physics modules there will be some general purpose software modules. One example is the `TIMING` module, which can time nested procedure calls, and report the timing results in various ways. Other examples can be general parallel linear solvers, error handlers, etc. These modules also have identifiers in all capital letters, but the identifier can have arbitrary length. The control module of the framework is also a software module, which controls and provides communication between the physics modules.

We note here that physics and software modules have nothing to do with Fortran 90 modules. Fortran 90 modules are simply collections of variables and procedures within a physics or software module.

1.4 File Names

File names of source files should reflect the content. If a source file contains a single procedure, it should be named the same as the intra-module name of the procedure with an extension specific to the language. For example a Fortran 90 source code file containing `subroutine calc_flux` should be named `calc_flux.f90`. If the source code is in Fortran 77 or C/C++, the extension is `.f` or `.c` respectively. If a source file contains a set of procedures, it should be named by a group name that describes the set of procedures, or by the main procedure in the group. If a source file containes a Fortran 90 module, it should be named accordingly, e.g. the file containing the ModMain module should be named `ModMain.f90`.

Source files will reside in separate directories for each module, therefore it is not necessary to include the module identifier into the file name.

1.5 Inter-Module Procedure Names

We distinguish intra-module procedure names and inter-module procedure names. Intra-module procedure names are used within a module, while inter procedure names are used in the framework. The conversion between these will be automatized, for example with the aid of a Perl script. This means that the modules can use arbitrary procedure names, still the name conflicts can be avoided.

The inter-module procedure names start with the physics/software module abreviation (see the list in the previous section) in all capitals, followed by an undescore, then followed by the intra-procedure name.

Here are some possible examples for the full inter-procedure names:

```
   name           ! meaning
-------------------------------------------------------------
   GM_calc_flux   ! Global magenetosphere flux calculation
   IH_get_var     ! inner heliosphere variable extraction
   MH_get_var     ! generic MHD variable extraction
   MH_update_b0   ! generic MHD B0 update
   TIMING_start   ! TIMING software module starts timing
```

Note that in the examples above the MHD module is not associated with a single physics module, thus it uses the module name `MH`. The control module will only call the subroutines for specific physics modules, e.g. for the inner heliosphere. These subroutines have to be all defined as simple interfaces for the generic `MH_*` subroutine.

1.6 Inter-Module Variable Names

Similarly to procedure names we distinguish intra-module and inter-module variable names. Intra-module variable names are used within each science module, and also in most software modules. The main exception is the control module, which must communicate with many other modules, and thus it uses inter-module variable names.

Inter-module variable names start with a prefix, which consists of the two character abbreviation of the physics module, followed by a type indicator, an optional dimension indicator, and an underscore. The prefix is then followed by the intra-module variable name. The type indicator can be the following:

b boolean

h 2-byte-integer

i 4-byte-integer

j 8-byte-integer

r real

d double precision real

c character

s string

e enumerated value

m F90 module

t type/structure

The dimension indicator is a positive integer number, which gives the number of indices for array variables. For scalar variables it is omitted. Some examples for prefixes:

```
GMr3_   - Global magnetosphere, real 3D array
IHi_    - Inner heliosphere, integer scalar
```

1.7 Intra-Module Procedure Names for BATSRUS

The intra-module procedure name consists of *procedure name parts* separated by undescores. A procedure name part starts with a lower case letter, followed by an arbitrary number of lower case letters and numbers. The use of lower case letters and undescores between the procedure name parts helps to distinguish procedure names from variable names, which use capitalization (see later). The intra-procedure name should describe the action done by the procedure, so it typically starts with a verb. Examples:

```
   name                ! meaning
-------------------------------------------------------------
   advance_impl        ! advance in time with implicit scheme
   calc_face_flux      ! calculate face fluxes
   set_b0              ! set the B0 magnetic field
   set_ics             ! set initial conditions
   read_restart_file   ! generic MHD B0 update
```

1.8 Intra-Module Variable Names for BATSRUS

The following suggestions were developed by G. Tóth and D. De Zeeuw for the magnetosphere module (the main part of BATSRUS). The original system was later improved by D. Chesney. The suggested naming system tries to resemble the current somewhat chaotic naming system to minimize the difficulties of the transition. More importantly we tried to create logical, unique, distinct and easy to read and write variable names. This naming system has been used in some recently written subroutines, and our limited experience shows that it works reasonably well, and definitely better than the current lack of system.

1.8.1 Name parts

Each intra-module variable name may consist of one or more *name parts*. All the parts must start with a capital letter and continue with lower case letters and numbers. There are two exceptions to this rule: (i) if the first name part consists of a single character, it should be lower case; (ii) the last name part may be one of the three scope descriptors `BLK`, `PE`, and `ALL`, which consists of all capital letters. In Fortran capitalization is ignored by the compiler, so mistakes in the capitalization have no effect on the correctness of the code. On the other hand consistent capitalization is essential to improve readability of the code. Examples of correct capitalizations:

```
i
iMax
UseConstrainB
DtALL
nBlockExpl
CflImpl
R2Min
InnerBcType
```

1.8.2 Indication of variable type

There are strict rules in this data naming standard to indicate the type (module, real, integer, logical, character string) of a variable. These rules are made as easy to read and write as possible:

Fortran 90 module names start with

```
Mod
```

All integer variable names must start with one of the following name parts:

```
i
j
k
l
m
n
Di
Dj
Dk
Dl
Dm
Dn
Min
Max
Int
```

All character and character string type variable names must start with any of

```
Name
Type
String
```

All logical variable names must start with one of

```
Do
Done
Is
Use
Used
Unused
```

Finally all real type variable names must start with a name part which was not listed in any of the above lists.

In addition to the consistent data names, we must make sure that the compiler checks the declaration and use of variables, thus the **implicit none** statement **must be used in every single procedure** in Fortran source code.

These rules should also help to make the order of the name parts less arbitrary. For example the minimum of the pressure should be named `pMin` and not `MinP`, because it is a real number which cannot start with the name part `Min` which is reserved for integers. Examples:

```
  UseConstrainB       - logical
  UnusedBlock         - logical
  nBlockUsed          - integer
  iProc               - integer
  RhoSwDim            - real
  xTest               - real
  TypeFlux            - character string
  NamePlotFile        - character string
```

1.8.3 Array variable names

Array variable names are distinguished from scalar variable names by the indication of indexes. The indexes are represented by an underscore followed by capital case letters at the end of the array variable name. Each capital letter represents one or more well defined indexes, and their order must be the same as the order of indexes in the declaration of the array variable. The following index abbreviations are defined:

```
name    typical range                 meaning
-----------------------------------------------
A       (1:nBlockALL)                 global blocks
B       (1:nBlock)                    local blocks
C       (1:nI,1:nJ,1:nK)              physical cells
D       (1:nDim)                      dimensions
E       (1:2*nDim)                    edges
F       (1:nI+1,1:nJ+1,1:nK+1)        faces
G       (-1:nI+2,-1:nJ+2,-1:nK+2)     ghost cells
I                                     general index (none of the others)
K       (1:nI+1,1:nJ+1,1:nK+1)        corners
N       (-1:1,-1:1,-1:1)              neighbors
P       (1:nProc)                     processors
Q       (1:4)                         quadrants
S       (1:2)                         sides
V       (1:nVar)                      (flow) variables
X       (1:nI+1,1:nJ,1:nK)            X faces
Y       (1:nI,1:nJ+1,1:nK)            Y faces
Z       (1:nI,1:nJ,1:nK+1)            Z faces
```

If there are many general indexes, one can use `_I3` instead of `_III`. Examples:

```
  Dx_B(:)             - real array indexed by blocks
  GradRho_C(:,:,:)    - real array indexed by cells
  B0x_XB(:,:,:,:)     - real array indexed by the X faces and blocks
  nRoot_D(:)          - integer array indexed by the 3 directions (x,y,z)
```

1.8.4 Indication of scope of variables

Some scalar variables or array elements refer to a single cell, of cell face, others refer to a whole block, processor, or even all the processors used by the module. For example in the current BATSRUS code `dt` is the time step for the whole simulation domain (i.e. all the processors), `dt_BLK(1)` is the smallest time step for the first block, `time_BLK(1,1,1,1)` is the time step for a single cell in the first block, while `TimeCell` is the time step for the cell being updated. The current names are somewhat arbitrary. In the new naming system the different scopes of a scalar variable or array element can be indicated by the following name parts:

```
  BLK     - one block
  PE      - one processing element
  ALL     - all the processing elements used by the module
```

These name parts are all capital, and they must be the last name part for scalars, and the last name part before the underscore for arrays. In the new naming system the above mentioned variables will become

```
New name    Old name     Meaning
-------------------------------------------------
DtALL       dt           global time step
DtBLK_B     dt_BLK       minimum time step for blocks
Dt_CB       time_BLK     local time step for cells
Dt          TimeCell     local time step for one cell
```

The last two names of the scalar `dt` and the four dimensional array `dt_CB` only differ in the array index abbreviations.

When the scope is not explicitly indicated, we assume the smallest scope that makes sense. For example `nBlock` is the number of blocks for a processor, while `nBlockALL` is the total number of blocks used by the module.

1.8.5 Named indexes

In the planned rewrite of BATSRUS, many variables will be collected into more general arrays. For example the conservative variables `rho_BLK, rhoUx_BLK,... E_BLK` will be put into a single array `Var_GVB`, i.e. the variable array is indexed by (ghost) cells, variable, and block number. Similarly the coordinate arrays `x_BLK, y_BLK, z_BLK` will be merged into `Xyz_GDB`, i.e. the coordinate array is indexed by ghost) cells, directions, and block number.

To make the variable and dimension indexes easier to read, *named indexes* are introduced. A named index is an integer constant (defined with the parameter statement in Fortran 90). The named index consists of the usual name parts followed by an underscore. The underscore is a reminder that named indexes have to do with arrays, and it also makes the syntax of named indexes different both from scalar and array variable names. Here is a list of named indexes (to be) introduced in BATSRUS:

```
name       value   meaning
-------------------------------------------------
x_         1       X index
y_         2       Y index
z_         3       Z index
Rho_       1       density index
RhoU_      1       momentum index
RhoUx_     2       X momentum index
RhoUy_     3       Y momentum index
RhoUz_     4       Z momentum index
B_         4       magnetic field index
Bx_        5       Bx index
By_        6       By index
Bz_        7       Bz index
e_         8       energy index
p_         9       pressure index
```

Examples of use:

```
  ! F_x[rho] = rho*U_x
  Flux_FDV(i,j,k,x_,Rho_)=Var_FV(i,j,k,RhoUx_)

  ! F_i[rho] = rhoU_i
  Flux_FDV(i,j,k,iDim,Rho_)=Var_FV(i,j,k,RhoU_+iDim)
```

Note the use of `RhoU_+iDim` which gives the index for the `iDim`-th component of the momentum. Both the flux and the variable arrays are face centered.

When velocity is used instead of momentum and/or spherical components are used instead of Cartesian, the following named indexes can be used:

name	value	meaning
U_	1	velocity index
Ux_	2	X velocity index
Uy_	3	Y velocity index
Uz_	4	Z velocity index
r_	1	Radial coordinate index
Phi_	2	Phi coordinate index
Theta_	3	Theta coordinate index
Ur_	2	Radial velocity index
Uphi_	3	Phi velocity index
Utheta_	4	Theta velocity index
Br_	2	Radial magnetic field index
Bphi_	3	Phi magnetic field index
Btheta_	4	Theta magnetic field index

1.8.6 Lists of name parts

General

name	meaning
c	constant (real, parameter ::)
d	difference
Dim	dimensional/dimension
Do	do something or not (logical)
Dt	time step, time interval
Dn	difference in time step index
i	index or index in direction X
j	2nd index or index in direction Y
k	3rd index or index in direction Z
Min	minimum of
Max	maximum (number) of
n	number of
Par	parameter
Read	read input file
Write	write output file
Save	save output file
Coeff	coefficient
Crit	criteria
Test	test of
Use	use a module/scheme (logical)

Basic flow variables

name	meaning

```
x         X direction/coordinate
y         Y direction/coordinate
z         Z direction/coordinate
xyz       coordinate in general
R         radial distance from origin
R2        radial distance from 2nd body
Time      time
Gamma     adiabatic index
Clight    speed of light
Rho       density
U         velocity
Ux        x component of velocity
Uy        y component of velocity
Uz        z component of velocity
P         pressure
T         temperature
E         energy
Ex        x component of electric field
Ey        y component of electric field
Ez        z component of electric field
B         magnetic field
Bx        x component of magnetic field
By        y component of magnetic field
Bz        z component of magnetic field
B0        split part of field
B0x       x component of split field
B0y       y component of split field
B0z       z component of split field
Jx        x component of current
Jy        y component of current
Jz        z component of current
Var       variable in general
Flux      flux
Source    source term
Sw        solar wind
Cme       coronal mass ejection
Arc       magnetic arcade in the solar corona
```

Discretization

```
name      meaning
-------------------------------------------------------
Inner     inner boundary at the surface of the body
Outer     outer boundary of the computational domain
Upstream  upstream boundary towards the sun
Iter      iterations for this run)
Step      time step for the whole (restarted) simulation
Stage     stage of the time discretization
Update    update of variables to the next time step
Expl      explicit time stepping
Impl      implicit time stepping
```

Bdf2	Backwad Difference Formula 2
Cfl	Courant-Friedrich-Lewy number (<1.0 for explicit)
Krylov	Krylov type iterative linear solver
Bicgstab	Bi-Conjugate Gradient STABilized scheme (Krylov type)
Gmres	General Minimum RESidual scheme (Krylov type)
Matvec	matrix vector multiplication in iterative solver
Newton	Newton iteration for non-linear problem
Schwarz	Schwarz type blockwise preconditioning
Precond	preconditioner to accelerate convergence
Mbilu	modified block incomplete lower-upper preconditioner
Rusanov	Rusanov's numerical flux function
Linde	Linde's numerical flux function
Roe	Roe's numerical flux function
Sokolov	Sokolov's numerical flux (artificial wind)
Grad	gradient
Limiter	function that limits slopes to avoid oscillations
Beta	Beta limiter
Minmod	Minmod slope limiter
Mc	Monotonized Central slope limiter
Eps	small number
Project	projection of B field to eliminate div B
Diff	diffusion of divergence of B
Powell	Powell's source term to advect div B
Constrain	constrained transport to conserve div B=0

AMR grid

name	meaning
Amr	Adaptive mesh refinement
Cell	cell
Coarsen	coarsening of grid
Face	face
Block	block
Nei	neighbour of
Lev	refinement level
Octree	octree of blocks
Prolong	prolongation of data for refinement
Refine	refinement of grid
Restrict	restriction of data for coarsening
Root	root of octree
Proc	processing element
Used	used block/node
Unused	unused block/node