Detailed Overview of the Plasma State Software

Presented at the EU-US Workshop on Software Technologies for Integrated Modeling, Dec. 1, 2010.

The Plasma State (PS) Software

- Native fortran-2003 implementation.
- Supplemented with detailed C++ facility for Plasma State object instantiation and access.
- Contents are defined from a specification file:
  - http://w3.pppl.gov/~dmccune/SWIM
    - Specification: plasma_state_spec.dat.
    - Early design documents.
    - Example Plasma State NetCDF files.
- Python script-generated source code.

PS Software Distribution

- http://w3.pppl.gov/NTCC
  - NTCC library module with dependencies
    - Build is laborious but we can provide technical support.
  - Works well with most Fortran-2003 compilers:
    - Pathscale, Intel, gfortran, Solaris, lf95 32-bit.
    - PGI (some routines require disabling of optimizer).
  - Now used by NUBEAM NTCC module.
- Code development: TRANSP svn repository
- Mirrored in: SWIM SciDAC svn repository

Plasma State in Fortran-2003

```
Program my_prog
  use plasma_state_mod  ! Definition, methods
  ! Also declares some instances: ps, psp, aux, …

  type (plasma_state) :: my_ps  ! User defined…

  call ps_init_user_state(my_ps, “my_ps”, ierr)
  if(ierr.ne.0)  <…handle error…>

  call ps_get_plasma_state(ierr, &
       filename=“my_ps.cdf”, state=my_ps)
  if(ierr.ne.0)  <…handle error…>

  write(6,*) ‘ #thermal species: ‘,my_ps%nspec_th
```

Plasma State in C++ [1]

```
void testCxx(int argc, char* argv[]) {

  int debug = getDebugLevel(argc, argv);
  // constructor
  PlasmaState ps("test", debug);

  // set number of thermal species
  ps.setData("nspec_th", 7);
  int nspec_th = ps.getData<int>("nspec_th");
  assert(nspec_th == 7);

  // allocate arrays
  ps.alloc();

  // store as file
  ps.store(“test.nc”);
```

See: *Exposing Fortran Derived Types to C and other languages*, A. Pletzer, D. McCune et al., CISE Jul/Aug 2008 (Vol. 10 No. 4).

PS Code Development History

- 2006-2007 –
  - Design discussions, SWIM project participants
    - Physicists, Programmers, CS experts
  - Version 1.xxx implementation by D. McCune
  - Version 2.xxx, major changes, designed late 2007.
- 2008-2010 –
  - Version 2.xxx implementation by D. McCune
  - Use in PTRANSP, SWIM, FACETs frameworks;
  - Use broadened to other projects e.g. TGYRO.
- Now at PSv2.029; ~1 FTE net labor investment.

Plasma State Object Contents

- Member elements are scalars and arrays of:
  - REAL(KIND=rspec), equivalent to REAL*8.
  - INTEGER.
  - CHARACTER*nnn – strings of various length.
- Flat structure, scalars and allocatable arrays:
  - All object members are primitive fortran types.
- Maximum element identifier length = 19
  - Alphabetic 1st character; then alphanumeric + "_"
  - 26*37**18 = 4.39*10**29 possible element names

Plasma State Contents (p. 2)

- C++ set/get method names have maximum length 19+13 = 32 characters.
- Semantic elements (constituted by one or more primitive PS object data elements):
  - Item lists (for example: list of neutral beams).
  - Species lists (for example: list of beam species).
  - Grids (for example: radial grid for neutral beam physics component).

Plasma State Sections

- Machine_Description
  - Time invariant, shot invariant for tokamak-epoch
- Shot_Configuration
  - Time invariant within a shot (e.g. species lists).
- Simulation_Init
  - Time invariant (e.g. grids & derived species lists).
- State_Data – non-gridded scalars and arrays.
- State_Profiles – arrays of gridded profiles.

Plasma State Physics Components

- Each data element is assigned to a physics component.
- List of components:
  - Plasma (pertaining to thermal species profiles)
  - EQ (pertaining to MHD equilibrium)
  - Heating components: NBI, IC, LH, EC
  - FUS (fusion products)
  - RAD (radiated power); GAS (neutral species)
  - RUNAWAY, LMHD, RIPPLE, ANOM (see spec.)

For Example: NBI Component

- Machine description:
  - List of neutral beams:
    - Names, detailed geometry, energy fraction tables.
- Shot configuration:
  - Injection species for each neutral beam.
- Simulation initialization:
  - Beam species list, derived from shot configuration.
  - Radial grid for NBI profile outputs.

NBI Component (p. 2)

- State Data
  - Neutral beam injector powers and voltages.
  - Injection fractions (full/half/third energy beam current fractions).
- State Profiles
  - Beam ion densities nb, and <Eperp>, <Epll>.
  - Main Heating: Pbe, Pbi, Pbth.
  - Main Torques: Tqbe, Tqbi, TqbJxB, Tqbth.
  - Particle source profiles, all thermal species.
  - Current drive, beam deposition halo profiles, etc.

PS: What’s in and What’s not

- Included in Plasma State: physics data shared between components:
  - E.g. neutral beam powers set by plasma model.
  - Profiles returned by NBI, used by plasma model.
- Not included:
  - Implementation specific controls:
    - E.g. NPTCLS for NUBEAM implementation of NBI.
  - Data specific to a single implementation only:
    - E.g. Monte Carlo code state as particle lists.
  - So far profiles of rank > 2 have not been used.

Item Lists in Specification File

```
# Coil/circuit description -- free boundary sim.
  L|pf_circuits circuit_name(ncircuits) ! PF circuits

  L|pf_coils coil_name(ncoils) ! Axisymmetric coils

  N  coil_in_circuit(ncoils) ! circuit to which
      ! each coil belongs (name must match exactly)

  R|units=m  Rloc_coil(ncoils) ! R, lower left corner
  R|units=m  Zloc_coil(ncoils) ! Z, lower left corner
      …etc…
```

- L – define list: CHARACTER*32 names & array dimension.
- N – CHARACTER*32 array of names.
- R – REAL*8 arrays or scalars with physical units (MKS & KeV).

Item Lists in Plasma State

List Label	Array of Names, Dimension	Component	Section
PF_circuits	Circuit_name(ncircuits)	EQ	Machine Descr.
PF_coils	Coil_name(ncoils)	EQ	Machine Descr.
Neutral_beams	NBI_src_name(nbeam)	NBI	Machine Descr.
ICRF_source	ICRF_src_name(nicrf_src)	IC	Machine Descr.
ECRF_source	ECRF_src_name(necrf_src)	EC	Machine Descr.
LHRF_source	LHRF_src_name(nlhrf_src)	LH	Machine Descr.
Gas_source	GS_name(ngsc0)	GAS	Machine Descr.
PS_moments*	PSmom_num(npsmom)	EQ	Sim. Init.
EQ_moments**	EQmom_num(neqmom)	EQ	Sim. Init.

*Neoclassical Pfirsch-Schlutter moments

**Fourier moments for a representation of core plasma flux surfaces

`ps%NBI_src_name(ps%nbeam)` – name of the last neutral beam in state "**`ps`**"

Species Lists in Specification File

```
# Main thermal plasma species list:
  S|thermal_specie S(0:nspec_th) ! All thermal species
    ! Index 0 for electrons
  S|fusion_ion SFUS(nspec_fusion) ! Fusion products
  S|RF_minority RFMIN(nspec_rfmin) ! RF minority ions

  S|beam_ion SNBI(nspec_beam) ! Beam species
    ! Derived from beam injector (nbeam) data

  S|specie  ALL(0:nspec_all)  ! All species
    ! Concatenation derived from primary species lists
```

- S – define species list: <root_name> & <array_dimension>
- CHARACTER*32 <root_name>_name(<array_dimension>)
- INTEGER <root_name>_type(<array_dimension>)
- REAL*8 q_<root_name>(<array_dimension>) – charge (C).
- REAL*8 m_<root_name>(<array_dimension>) – mass (kg).

Using Species List Data

```
! The plasma_state_mod module defines parameters:
  ! Mass of proton (KG)
  REAL(KIND=rspec), parameter :: ps_mp = 1.6726e-27_rspec
  ! Unitary charge (C)
  REAL(KIND=rspec), parameter :: ps_xe = 1.6022e-19_rspec

! For data in state object "ps": last ion in thermal list:
  ! mass divided by proton mass
  A = ps%m_s(ps%nspec_th)/ps_mp
  ! Atomic charge
  Zatom = ps%Qatom_s(ps%nspec_th)/ps_xe
  ! Ionic charge (Zion = Zatom for fully stripped ion):
  Zion = ps%q_s(ps%nspec_th)/ps_xe

  do i=1, ps%nspec_th
    write(6,*) i, ps%Qatom_s(i)/ps_xe, ps%q_s(i)/ps_xe, &
              ps%m_s(i)/ps_mp
  enddo
```

Species Lists in Plasma State

List Label	Array Name Root, Dimension	Component	Section
Thermal_specie	S(0:nspec_th)	PLASMA	Shot Config.
Fusion_ion	SFUS(nspec_fusion)	FUS	Shot Config.
RF_minority	RFMIN(nspec_rfmin)	IC	Shot Config.
Beam_ion	SNBI(nspec_beam)	NBI	Sim. Init.
Specie	ALL(0:nspec_all)*	PLASMA	Sim. Init.
Thermal_specie	SA(0:nspec_tha)**	PLASMA	Sim. Init.
Specie	ALLA(0:nspec_alla)***	PLASMA	Sim. Init.
Neutral_gas	SGAS(nspec_gas)	GAS	Sim. Init.
Impurity_atoms	SIMP0(nspec_imp0)	GAS	Sim. Init.

* all-species list: all thermal species & all fast ions, combined in single list.

** abridged thermal species list: impurities merged.

*** abridged thermal species & all fast ions, combined in single list.

`ps%SA_name(ps%nspec_tha)` – name of last thermal ion specie, abridged list.

Grids in the Plasma State

Grid Array Name, Dimension	Component	Section
rho(nrho)*	PLASMA	Sim. Init.
rho_eq(nrho_eq)	EQ	Sim. Init.
th_eq(nth_eq)**	EQ	Sim. Init.
R_grid(nR)	EQ	Sim. Init.
Z_grid(nZ)	EQ	Sim. Init.
rho_eq_geo(nrho_eq_geo)	EQ	Sim. Init.
rho_nbi(nrho_nbi)	NBI	Sim. Init.
rho_fus(nrho_fus)	FUS	Sim. Init.
(etc., etc., etc.)	All components	Sim. Init.

* “rho” radial grids: sqrt(<normalized-toroidal-flux>), range [0.00:1.00].

** “th” poloidal angle grid, range [0.00:2*pi] or [-pi:+pi].

All grids are aligned with *boundaries* of numerical zones, covering the *entire* range of their respective coordinate domains.

Use of PS in Simulation

- Initialization:
  - Driver code sets up item lists by reading machine description file (an ascii namelist).
  - Driver code sets up species lists for simulation.
  - Components each set up their own grids.
  - Plasma State supports partial allocation, allowing for distributed multi-step initialization strategy.
- Time dependent use:
  - Components update data in time loop.

Plasma State Array Allocation

- Procedure:
  - Set array dimension sizes (e.g. `ps%nrho_nbi = 21`).
  - Call module routine:
    - `CALL ps_alloc_plasma_state(ierr, state=ps)`
      - `Set grid values ps%rho_nbi(1:ps%nrho_nbi) = …`
    - Unallocated arrays with ***all*** dimensions defined (i.e. greater than 0) are allocated by call.
    - Each array can only be allocated ***once*** in the history of a plasma state object.
    - Dynamic re-gridding can be done but requires:
      - Creation of a new state object; copying & interpolation of data.

PS Interpolation Services

- Components provide data on their native grids.
- Interpolation typically required for use.
- Plasma State definition provides "recommended" interpolation method for each defined profile:
  - Spline, Hermite, piecewise linear, zone step functions
  - Conservative "rezoning" of profiles:
    - For densities & sources conserve #, #/sec, Watts, …
    - For temperatures conserve volume integrated n*T.
- Interpolation libraries: xplasma, pspline (NTCC).

Profile Interpolation by Rezoning

```
! Plasma State “ps” has fine PLASMA grid ps%nrho = 101
! and coarse NBI grid ps%nrho_nbi = 21
  use plasma_state_mod ! Interpolation data tags id_<name>(…)
  ! Test arrays:
  real*8, dimension(:), allocatable :: my_te, my_pbi

  ! Allocate arrays with zone-centered orientation
  allocate(my_te(ps%nrho_nbi-1),my_pbi(ps%nrho-1))

  ! Fine-to-coarse rezone (ne*Te sum conserved):
  call ps_rho_rezone(ps%rho_nbi, ps%id_Ts(0), my_te, ierr, &
             state=ps)

  ! Coarse-to-fine rezone with smoothing to suppress step
  ! function structure (PBI sum conserved, output in W/zone,
  ! local radial shifts up to ½ width of coarse zones).
  call ps_rho_rezone(ps%rho, ps%id_pbi, my_pbi, ierr, &
             state=ps, nonorm=.TRUE., zonesmoo=.TRUE.)
```

PS I/O Services

- Ps_get_Plasma_State – read all from NetCDF
- Ps_store_Plasma_state – write all to NetCDF
- Ps_read_update_file – read a Plasma State update, e.g. data from a separate component.
- Ps_write_update_file – write an update: changed elements only.
- Interpolation data *updated* on each call.
- All I/O subroutines as well as object definitions written and updated by Python code generator.

PS Version Compatibility

- Current released SWIM version: 2.029.
- All version 2.xxx states compatible
  - Code linked to newer PS software can read old version state file; some data items missing.
  - Code linked to older PS software can read new version state file; some data items not used.
- Version interoperability maintained by the Python code generator.
- So: version updates are relatively painless.

PS Definition Update Procedure

- Edit the specification file.
- Run the Python code generator.
- Run compatibility tests.
- Commit

Current Utilization of Plasma State

Drivers:
TRANSP/PTRANSP
SWIM SciDAC
FACETS SciDAC

Plasma State

FSP Components:
NUBEAM, GENRAY
CQL3D, TORIC, …

TRANSP Data Archives (MDS+)

trxpl executable*

*or subroutine library

Single Plasma State

Plasma State time series

First Principles Slice Analysis
- MHD stability (PEST, M3D…)
- Transport (GYRO, GTC…)
- Diagnostic simulation, etc.

Advanced time dependent simulation
- Now: TSC, SWIM IPS
- Soon: FACETS
- Possible: CPES, FSP…

Successes

- Data standardization facilities sharing of major physics components:
  - E.g. NBI & FUS (implemented by NUBEAM), the same code, used by TRANSP/PTRANSP, SWIM, FACETS.
    - Workstations & small clusters, serial, small scale MPI.
    - Supercomputers, MPI to low 1000s of processors.
- Data standardization facilitates verification of component implementations:
  - E.g. AORSA & TORIC comparisons in IC component.

Performance Considerations

- Plasma State I/O is serial overhead.
  - But Plasma State aggregate sizes are usually small;
  - ~500 scalar lists and low rank profile elements;
  - 0.5-5Mbytes as NetCDF, modestly larger memory footprint due to interpolation data;
  - Not a limiting factor in present day applications.
- But this could change quickly if PS is ever extended to include rank 3 or higher profiles.
  - Domain decompositions not yet considered.

Advanced Techniques (1)

- Create Plasma State with TRANSP profiles but high resolution JSOLVER MHD equilibrium:
  - Extract TRANSP state which includes low resolution MHD equilibrium (EQ);
  - Selective copy to new state object, omitting EQ;
    - **`CALL ps_copy_plasma_state(…)`** & use **`cclist(:)`** control.
  - Use JSOLVER, compute high resolution EQ;
  - Allocate and write EQ in the copied state object; write to output file.

Advanced Techniques (2)

- Weighted average of two state objects, creating a 3rd state object.
  - Read or create 2 state objects with congruent dimensioning
    - E.g. as taken from TRANSP archives via “trxpl”.
  - Merge the two states into a 3rd state with indicated weighting:
    - `CALL ps_merge_plasma_state(weight1, ps1, ps2, &`
    - `          new_state = ps3)`
    - **Result:** `ps3 = weight1*ps1 + (1-weight1)*ps2`
  - Use e.g. for time interpolation.

Concluding Remarks

- A detailed overview of Plasma State was presented.
- The software has been useful for integrating components
  - Context: 1.5d transport simulation.
- The software has been useful for sharing data:
  - Experimental data in TRANSP archives made available to theory codes: TGYRO, TSC, SciDACs…
- So far only used for "small" data.
  - Gridded data elements of rank at most 2.