IPDv4
Interoperable Physics Driver

Weekly FV3GFS Technical Meeting
13 March, 2017

IPDv4.0

IPD_typdefs.F90

container definitions

IPD_driver.F90

interface to physics routines

IPD_typedefs

Defines containers to abstract the the specific variables needed by a given physics suite

```
type(IPD_control_type) ::                IPD_Control
type(IPD_data_type) ::                   IPD_Data
type(IPD_diag_type), allocatable ::      IPD_Diag(:)
type(IPD_restart_type) ::                IPD_Restart
```

IPD Data

```
type IPD_data_type
   public
   type(statein_type)  :: Statein
   type(stateout_type) :: Stateout
   type(sfcprop_type)  :: Sfcprop
   type(coupling_type) :: Coupling
   type(grid_type)     :: Grid
   type(tbd_type)      :: Tbd
   type(cldprop_type)  :: Cldprop
   type(radtend_type)  :: Radtend
   type(diag_type)     :: Intdiag
 end type IPD_data_type
```

Subtypes defined by the specific physics library

IPD Restart

```
type IPD_restart_type
   public
   integer                :: num2d
   integer                :: num3d
   character(len=32), allocatable :: name2d(:)
   character(len=32), allocatable :: name3d(:)
   type(var_subtype), allocatable :: data(:,:)
end type IPD_restart_type


type var_subtype
   real(kind=kind_phys), pointer :: fld2d(:)   => null()
   real(kind=kind_phys), pointer :: fld23(:,:) => null()
end type var_subtype
```

IPD Diag

```
type IPD_diag_type
   public
   character(len=32)      :: name
   character(len=32)      :: output_name
   character(len=32)      :: mod_name
   character(len=32)      :: file_name
   character(len=128)     :: desc
   character(len=32)      :: unit
   character(len=32)      :: type_stat_proc
   character(len=32)      :: level_type
   integer                :: level
   real(kind=kind_phys)   :: cnvfac
   real(kind=kind_phys)   :: zhour
   real(kind=kind_phys)   :: fcst_hour
   type(var_subtype), allocatable :: data(:)
end type IPD_diag_type
```

IPD Calls

Call IPD_initialize (IPD_Control, IPD_Data, IPD_Diag,\
IPD_Restart, Init_parm)

call IPD_<stage> (IPD_Control, IPD_Data, IPD_Diag,\
IPD_Restart)

where stage is:

- setup_step
- radiation_step[N]
- physics_step[N]

Using IPD

```
Integration Loop:
   call atmos_dynamics ()

   call populate_state (IPD_Data)

   call IPD_setup_step (IPD_Control, IPD_Data, IPD_Diag,  &
                        IPD_Restart)

   call IPD_radiation_step1 (IPD_Control, IPD_Data, &
                             IPD_Diag, IPD_Restart)

   call IPD_physics_step1 (IPD_Control, IPD_Data,  &
                           IPD_Diag, IPD_Restart)

   call IPD_physics_step2 (IPD_Control, IPD_Data,  &
                           IPD_Diag, IPD_Restart)

   call update_prognostic_state (IPD_Data)

   call output_diagnostics (IPD_Diag)
End Integration Loop
```

IPD Layer

```
call initialize (IPD_Control,          IPD_Data%Statein, &
                 IPD_Data%Stateout,    IPD_Data%Sfcprop,&
                 IPD_Data%Coupling,    IPD_Data%Grid,    &
                 IPD_data%Tbd,         IPD_Data%Cldprop,&
                 IPD_Data%Radtend,     IPD_Data%IntDiag, &
                 IPD_Diag,        IPD_Restart,     Init_parm)
```

```
call <stage>    (IPD_Control,          IPD_Data%Statein, &
                 IPD_Data%Stateout,    IPD_Data%Sfcprop,&
                 IPD_Data%Coupling,    IPD_Data%Grid,    &
                 IPD_data%Tbd,         IPD_Data%Cldprop,&
                 IPD_Data%Radtend,     IPD_Data%IntDiag)
```

Where stage is: time_vary_step, radiation_step1, physics_step1, physics_step2

Library Abstraction Layer

Associate via module use statement:

```
initialize          =>    GFS_initialize
time_vary_step      =>    GFS_time_vary_step
radiation_step1     =>    GFS_radiation_driver
physics_step1       =>    GFS_physics_driver
physics_step2       =>    GFS_stochastic_physics

public initialize, time_vary_step,      &
       radiation_step1,                 &
       physics_step1, physics_step2
```

Library Abstraction Layer

Same concept for the typedefs:

Statein	=>	GFS_statein_type
Stateout	=>	GFS_stateout_type
Sfcprop	=>	GFS_sfcprop_type
Coupling	=>	GFS_coupling_type
Grid	=>	GFS_grid_type
Tbd	=>	GFS_tbd_type
CldProp	=>	GFS_cldprop_type
Radtend	=>	GFS_radtend_type
IntDiag	=>	GFS_diag_type

Physics: Initialization

Beyond the the standard initialization procedures, responsible for:

- populating the IPD control container
- allocating all IPD_Data sub-elements
- allocating/populating the IPD diagnostic container
- populating the IPD restart container

Physics: Control type

Defines all physics/radiation control parameters in one container

Given default values (exception: derived from others)

Under namelist control (emerging technologies: JSON)

Includes dynamic integration quantities

current date/time-step counter
parameterization triggers
solar quantities
etc.

Physics: Init Container

Non-physics data needed to initialize the physics

Specific to a given suite

```
type init_type
   public
   integer :: me
   integer :: master
   integer :: nx
   integer :: ny
   integer :: levs
   integer :: gnx
   integer :: gny
   integer :: nlunit
   integer :: dt_dycore
   integer :: dt_phys
   integer :: bdat(8)
   integer :: cdat(8)
```

Physics: init_type (cont'd)

```
  !--- blocking data
  integer, pointer :: blksz(:)
  !--- ak/bk for pressure level calculations
  integer, pointer :: ak(:)
  integer, pointer :: bk(:)
  !--- grid metrics
  real(kind=kind_phys), pointer :: xlon(:,:)
  real(kind=kind_phys), pointer :: xlat(:,:)
  real(kind=kind_phys), pointer :: area(:,:)
  character(len=32) :: tracer_names(:)
  character(len=65) :: fn_nml
end type init_type
```

Physics Suite: Initialization

Example of container initialization

```
call Statein %create (Init_parm%size, IPD_Control)
call Stateout%create (Init_parm%size, IPD_Control)
call Sfcprop %create (Init_parm%size, IPD_Control)
call Coupling%create (Init_parm%size, IPD_Control)
call Grid    %create (Init_parm%size, IPD_Control)
call Tbd     %create (Init_parm%size, IPD_Control)
call Cldprop %create (Init_parm%size, IPD_Control)
call Radtend %create (Init_parm%size, IPD_Control)
!--- internal representation of diagnostics
call Diag    %create (Init_parm%size, IPD_Control)
```

Blocking

Defines containers to abstract the the specific variables needed by a given physics suite

```
type(IPD_control_type) ::                 IPD_Control
type(IPD_data_type), allocatable ::       IPD_Data(:)
type(IPD_diag_type), allocatable ::       IPD_Diag(:)
type(IPD_restart_type) ::                 IPD_Restart

allocate (IPD_Data(nblks))
```

Blocking

```
do nb = 1,size(Init_parm%blksz)
 call Statein (nb)%create (Init_parm%blksz(nb), IPD_Control)
 call Stateout(nb)%create (Init_parm%blksz(nb), IPD_Control)
 call Sfcprop (nb)%create (Init_parm%blksz(nb), IPD_Control)
 call Coupling(nb)%create (Init_parm%blksz(nb), IPD_Control)
 call Grid    (nb)%create (Init_parm%blksz(nb), IPD_Control)
 call Tbd     (nb)%create (Init_parm%blksz(nb), IPD_Control)
 call Cldprop (nb)%create (Init_parm%blksz(nb), IPD_Control)
 call Radtend (nb)%create (Init_parm%blksz(nb), IPD_Control)
      !--- internal representation of diagnostics
 call Diag    (nb)%create (Init_parm%blksz(nb), IPD_Control)
enddo
```

Blocking → Threading

Serial

```
    call IPD_physics_step1 (IPD_Control, IPD_Data, &
                            IPD_Diag, IPD_Restart)
```

Becomes

```
!$OMP parallel do default (none) &
!$OMP             shared   (Atm_block, IPD_Control) &
!$OMP             shared   (IPD_Data, IPD_Diag, IPD_Restart) &
!$OMP             private  (nb)
      do nb = 1,Atm_block%nblks
        call IPD_physics_step1 (IPD_Control, IPD_Data(nb), &
                                IPD_Diag, IPD_Restart)
      enddo
```

GFS physics library

IPD:

IPD_typedefs.F90

IPD_driver.F90

GFS physics:

GFS_typedefs.F90

GFS_driver.F90

GFS_diagnostics.F90

GFS_restart.F90

Modeling System:

<system>_io.F90

Init_type and call to populate Statein:

may need to be programmed

GFS Physics

gbphys → GFS_physics_driver

grrad → GFS_radiation_driver

First step in reducing complexity of GFS physics

Now accepts IPD argument list and abstracts fields/variables at the scheme/parameterization level

Next steps: clean up the logic used to choose active schemes within the physics driver

GFDL-AM4 physics library

IPD:

IPD_typedefs.F90

IPD_driver.F90

GFDL-AM4 physics:

GFDL_AM4_typedefs.F90

GFDL_AM4_driver.F90

GFDL_AM4_diagnostics.F90

GFDL_AM4_restart.F90

Modeling System:

<system>_io.F90

Init_type and call to populate Statein

may need to be programmed

CCPP physics library

IPD:

IPD_typedefs.F90

IPD_driver.F90

CCPP physics:

CCPP_typedefs.F90

CCPP_driver.F90

CCPP_diagnostics.F90

CCPP_restart.F90

Modeling System:

<system>_io.F90

Init_type and all to populate Statein:

may need to be programmed