```
module output_manager_core

   use iso_fortran_env, only: error_unit, output_unit

   use field_manager
   use yaml_settings

   implicit none

   public type_output_variable_settings,type_output_item,type_output_field, type_file, write_time_string, read_time_st
ring, host, type_host
   public type_base_output_field, type_base_operator, wrap_field

   private

   integer,parameter,public :: max_path = 256

   integer,parameter,public :: time_method_none          = 0  ! time-independent variable
   integer,parameter,public :: time_method_instantaneous = 1
   integer,parameter,public :: time_method_mean          = 2
   integer,parameter,public :: time_method_integrated    = 3

   integer,parameter,public :: time_unit_none   = 0
   integer,parameter,public :: time_unit_second = 1
   integer,parameter,public :: time_unit_hour   = 2
   integer,parameter,public :: time_unit_day    = 3
   integer,parameter,public :: time_unit_month  = 4
   integer,parameter,public :: time_unit_year   = 5
   integer,parameter,public :: time_unit_dt     = 6
   integer,parameter,public :: time_from_list   = 7

   integer,parameter,public :: rk = kind(_ONE_)

   type,abstract :: type_host
   contains
      procedure (host_julian_day),deferred :: julian_day
      procedure (host_calendar_date),deferred :: calendar_date
      procedure :: fatal_error => host_fatal_error
      procedure :: log_message => host_log_message
   end type

   abstract interface
      subroutine host_julian_day(self,yyyy,mm,dd,julian)
         import type_host
         class (type_host), intent(in) :: self
         integer, intent(in)  :: yyyy,mm,dd
         integer, intent(out) :: julian
      end subroutine
   end interface

   abstract interface
      subroutine host_calendar_date(self,julian,yyyy,mm,dd)
         import type_host
         class (type_host), intent(in) :: self
         integer, intent(in)  :: julian
         integer, intent(out) :: yyyy,mm,dd
      end subroutine
   end interface

   type type_output_variable_settings
      integer :: time_method = time_method_instantaneous
      class (type_base_operator), pointer :: final_operator => null()
   contains
      procedure :: initialize => output_variable_settings_initialize
      procedure :: finalize   => output_variable_settings_finalize
   end type

   type type_output_item
      class (type_output_variable_settings), pointer :: settings => null()
      character(len=string_length)         :: name = ''
      character(len=string_length)         :: prefix = ''
      character(len=string_length)         :: postfix = ''
      integer                              :: output_level = output_level_default
      class (type_category_node),  pointer :: category => null()
      type (type_field),           pointer :: field => null()
      type (type_output_item),     pointer :: next => null()
   end type

   type type_output_field_pointer
      class (type_base_output_field), pointer :: p => null()
   end type

   type type_base_output_field
      class (type_output_variable_settings), pointer :: settings => null()
      character(len=string_length)                   :: output_name = ''
      logical                                        :: is_coordinate = .false.
      class (type_category_node),  pointer           :: category => null()
      type (type_nd_data_pointer)                    :: data
      type (type_output_field_pointer), allocatable  :: coordinates(:)
      class (type_base_output_field),        pointer :: next => null()
   contains
      procedure :: new_data         => base_field_new_data
      procedure :: before_save      => base_field_before_save
      procedure :: flag_as_required => base_field_flag_as_required
      procedure :: get_metadata     => base_field_get_metadata
      procedure :: get_field        => base_field_get_field
      procedure :: finalize         => base_field_finalize
   end type type_base_output_field
```

```
 98
 99    type, extends(type_base_output_field) :: type_output_field
100       type (type_field), pointer :: source   => null()
101       logical,           pointer :: used_now => null()
102    contains
103       procedure :: flag_as_required => field_flag_as_required
104       procedure :: get_metadata     => field_get_metadata
105    end type type_output_field
106
107    type type_file
108       type (type_field_manager),    pointer :: field_manager   => null()
109       character(len=max_path)                 :: path           = ''
110       character(len=max_path)                 :: postfix        = ''
111       character(len=string_length)            :: title          = ''
112       type (type_attributes)                  :: attributes
113       integer                                 :: time_unit      = time_unit_none
114       integer                                 :: time_step      = 0
115       integer                                 :: first_index    = 0
116       integer                                 :: next_julian    = -1
117       integer                                 :: next_seconds   = -1
118       integer                                 :: first_julian   = -1
119       integer                                 :: first_seconds  = -1
120       integer                                 :: last_julian    = huge(1)
121       integer                                 :: last_seconds   = 0
122       type (type_output_item),       pointer :: first_item    => null()
123       class (type_base_output_field),pointer :: first_field   => null()
124       class (type_file),             pointer :: next          => null()
125    contains
126       procedure :: configure
127       procedure :: initialize
128       procedure :: save
129       procedure :: finalize
130       procedure :: create_settings
131       procedure :: is_dimension_used
132       procedure :: append_item
133    end type type_file
134
135    type type_base_operator
136       integer                               :: refcount = 1
137       class (type_base_operator), pointer :: previous => null()
138    contains
139       procedure :: configure   => operator_configure
140       procedure :: apply       => operator_apply
141       procedure :: dereference => operator_dereference
142       procedure :: finalize    => operator_finalize
143       procedure :: apply_all   => operator_apply_all
144    end type
145
146    class (type_host),pointer,save :: host => null()
147
148 contains
149
150    recursive subroutine base_field_flag_as_required(self, required)
151       class (type_base_output_field), intent(inout) :: self
152       logical, intent(in) :: required
153    end subroutine
154
155    recursive subroutine base_field_get_metadata(self, long_name, units, dimensions, minimum, maximum, fill_value, stan
    dard_name, path, attributes)
156       class (type_base_output_field), intent(in) :: self
157       character(len=:), allocatable, intent(out), optional :: long_name, units, standard_name, path
158       type (type_dimension_pointer), allocatable, intent(out), optional :: dimensions(:)
159       real(rk), intent(out), optional :: minimum, maximum, fill_value
160       type (type_attributes), intent(out), optional :: attributes
161       if (present(dimensions)) allocate(dimensions(0))
162    end subroutine
163
164    recursive subroutine base_field_new_data(self)
165       class (type_base_output_field), intent(inout) :: self
166    end subroutine
167
168    recursive subroutine base_field_before_save(self)
169       class (type_base_output_field), intent(inout) :: self
170    end subroutine
171
172    recursive function base_field_get_field(self, field) result(output_field)
173       class (type_base_output_field), intent(in) :: self
174       type (type_field), target                  :: field
175       class (type_base_output_field), pointer    :: output_field
176       output_field => wrap_field(field, .false.)
177    end function
178
179    recursive subroutine base_field_finalize(self)
180       class (type_base_output_field), intent(inout) :: self
181       if (associated(self%settings)) then
182          call self%settings%finalize()
183          deallocate(self%settings)
184       end if
185    end subroutine
186
187    function wrap_field(field, allow_unregistered) result(output_field)
188       type (type_field), target          :: field
189       logical, intent(in)                :: allow_unregistered
190       class (type_output_field), pointer :: output_field
191       output_field => null()
192       select case (field%status)
193       case (status_not_registered)
194          if (allow_unregistered) then
```

```
195            call host%log_message('WARNING: output field "'//trim(field%name)//'" is skipped because it has not been r
   egistered with field manager.')
196         else
197            call host%fatal_error('wrap_field', 'Requested output field "'//trim(field%name)//'" has not been register
   ed with field manager.')
198         end if
199      case (status_registered_no_data)
200         call host%fatal_error('wrap_field', 'Data for requested field "'//trim(field%name)//'" have not been provided
    to field manager.')
201      case default
202         allocate(output_field)
203         output_field%source => field
204         output_field%data = output_field%source%data
205         output_field%output_name = trim(field%name)
206         output_field%used_now => field%used_now
207      end select
208   end function
209
210   recursive subroutine field_flag_as_required(self, required)
211      class (type_output_field), intent(inout) :: self
212      logical, intent(in) :: required
213      if (associated(self%used_now) .and. required) self%used_now = .true.
214   end subroutine
215
216   recursive subroutine field_get_metadata(self, long_name, units, dimensions, minimum, maximum, fill_value, standard_
   name, path, attributes)
217      class (type_output_field), intent(in) :: self
218      character(len=:), allocatable, intent(out), optional :: long_name, units, standard_name, path
219      type (type_dimension_pointer), allocatable, intent(out), optional :: dimensions(:)
220      type (type_attributes), intent(out), optional :: attributes
221      real(rk), intent(out), optional :: minimum, maximum, fill_value
222
223      if (self%source%status == status_not_registered) then
224         if (present(dimensions)) allocate(dimensions(0))
225         return
226      end if
227      if (present(long_name)) long_name = trim(self%source%long_name)
228      if (present(units)) units = trim(self%source%units)
229      if (present(dimensions)) then
230         allocate(dimensions(size(self%source%dimensions)))
231         dimensions(:) = self%source%dimensions(:)
232      end if
233      if (present(minimum)) minimum = self%source%minimum
234      if (present(maximum)) maximum = self%source%maximum
235      if (present(fill_value)) fill_value = self%source%fill_value
236      if (present(standard_name) .and. self%source%standard_name /= '') standard_name = trim(self%source%standard_name
   )
237      if (present(path) .and. associated(self%source%category)) path = trim(self%source%category%get_path())
238      if (present(attributes)) call attributes%update(self%source%attributes)
239   end subroutine
240
241   subroutine configure(self, settings)
242      class (type_file),     intent(inout) :: self
243      class (type_settings), intent(inout) :: settings
244   end subroutine
245
246   subroutine initialize(self)
247      class (type_file),intent(inout) :: self
248      stop 'output_manager_core:initialize not implemented'
249   end subroutine
250
251   function create_settings(self) result(settings)
252      class (type_file),intent(inout) :: self
253      class (type_output_variable_settings), pointer :: settings
254      allocate(settings)
255   end function create_settings
256
257   subroutine save(self,julianday,secondsofday,microseconds)
258      class (type_file),intent(inout) :: self
259      integer,          intent(in)    :: julianday,secondsofday,microseconds
260      stop 'output_manager_core:save not implemented'
261   end subroutine
262
263   subroutine finalize(self)
264      class (type_file), intent(inout) :: self
265
266      class (type_base_output_field), pointer :: current, next
267
268      current => self%first_field
269      do while (associated(current))
270         next => current%next
271         call current%finalize()
272         deallocate(current)
273         current => next
274      end do
275   end subroutine
276
277   subroutine write_time_string(jul,secs,timestr)
278      integer,         intent(in)  :: jul,secs
279      character(len=*),intent(out) :: timestr
280
281      integer :: ss,min,hh,dd,mm,yy
282
283      hh   = secs/3600
284      min  = (secs-hh*3600)/60
285      ss   = secs - 3600*hh - 60*min
286
287      call host%calendar_date(jul,yy,mm,dd)
```

```
288
289       write(timestr,'(i4.4,a1,i2.2,a1,i2.2,1x,i2.2,a1,i2.2,a1,i2.2)')  &
290                             yy,'-',mm,'-',dd,hh,':',min,':',ss
291    end subroutine write_time_string
292
293    subroutine read_time_string(timestr,jul,secs,success)
294       character(len=19)    :: timestr
295       integer, intent(out) :: jul,secs
296       logical, intent(out) :: success
297
298       integer   :: ios
299       character :: c1,c2,c3,c4
300       integer   :: yy,mm,dd,hh,min,ss
301
302       read(timestr,'(i4,a1,i2,a1,i2,1x,i2,a1,i2,a1,i2)',iostat=ios)  &
303                             yy,c1,mm,c2,dd,hh,c3,min,c4,ss
304       success = ios == 0
305       if (ios==0) then
306          call host%julian_day(yy,mm,dd,jul)
307          secs = 3600*hh + 60*min + ss
308       end if
309    end subroutine read_time_string
310
311    subroutine host_fatal_error(self,location,error)
312       class (type_host), intent(in) :: self
313       character(len=*),  intent(in) :: location,error
314
315       write (error_unit,*) trim(location)//': '//trim(error)
316       stop 1
317    end subroutine
318
319    subroutine host_log_message(self,message)
320       class (type_host), intent(in) :: self
321       character(len=*),  intent(in) :: message
322
323       write (output_unit,*) trim(message)
324    end subroutine
325
326    logical function is_dimension_used(self,dim)
327       class (type_file),intent(inout) :: self
328       type (type_dimension), target   :: dim
329
330       class (type_base_output_field),pointer :: output_field
331       type (type_dimension_pointer), allocatable :: dimensions(:)
332       integer :: i
333
334       is_dimension_used = .true.
335       output_field => self%first_field
336       do while (associated(output_field))
337          call output_field%get_metadata(dimensions=dimensions)
338          do i=1,size(dimensions)
339             if (associated(dimensions(i)%p,dim)) return
340          end do
341          output_field => output_field%next
342       end do
343       is_dimension_used = .false.
344    end function is_dimension_used
345
346
347    subroutine append_item(self, item)
348       class (type_file),intent(inout) :: self
349       type (type_output_item), target :: item
350
351       ! Select this category for output in the field manager.
352       if (.not.associated(item%settings)) item%settings => self%create_settings()
353       if (.not. associated(item%field)) item%category => self%field_manager%select_category_for_output(item%name, item
    %output_level)
354
355       ! Prepend to list of output categories.
356       item%next => self%first_item
357       self%first_item => item
358    end subroutine append_item
359
360    subroutine output_variable_settings_initialize(self, settings, parent)
361       class (type_output_variable_settings), intent(inout)        :: self
362       class (type_settings),                 intent(inout)        :: settings
363       class (type_output_variable_settings), intent(in), optional :: parent
364
365       integer                                :: display
366       class (type_base_operator), pointer :: op
367
368       display = display_normal
369       if (present(parent)) then
370          self%time_method = parent%time_method
371          self%final_operator => parent%final_operator
372          display = display_advanced
373          op => parent%final_operator
374          do while (associated(op))
375             op%refcount = op%refcount + 1
376             op => op%previous
377          end do
378       end if
379
380       self%time_method = settings%get_integer('time_method', 'treatment of time dimension', options=(/option(time_meth
    od_mean, 'mean', 'mean'), &
381          option(time_method_instantaneous, 'instantaneous', 'point'), option(time_method_integrated, 'integrated', 'in
    tegrated')/), default=self%time_method, display=display)
382    end subroutine output_variable_settings_initialize
```

```
383
384    recursive subroutine output_variable_settings_finalize(self)
385       class (type_output_variable_settings), intent(inout) :: self
386       if (associated(self%final_operator)) then
387          if (self%final_operator%dereference()) deallocate(self%final_operator)
388       end if
389    end subroutine
390
391    subroutine operator_configure(self, settings, field_manager)
392       class (type_base_operator), target, intent(inout) :: self
393       class (type_settings),              intent(inout) :: settings
394       type (type_field_manager),          intent(inout) :: field_manager
395    end subroutine
396
397    function operator_apply(self, source) result(output_field)
398       class (type_base_operator), intent(inout), target :: self
399       class (type_base_output_field), target            :: source
400       class (type_base_output_field), pointer           :: output_field
401       output_field => source
402    end function
403
404    recursive function operator_apply_all(self, source) result(output_field)
405       class (type_base_operator), intent(inout), target :: self
406       class (type_base_output_field), target            :: source
407       class (type_base_output_field), pointer           :: output_field
408       output_field => source
409       if (associated(self%previous)) output_field => self%previous%apply_all(output_field)
410       if (associated(output_field)) output_field => self%apply(output_field)
411    end function
412
413    logical recursive function operator_dereference(self)
414       class (type_base_operator), intent(inout) :: self
415       self%refcount = self%refcount - 1
416       operator_dereference = self%refcount == 0
417       if (operator_dereference) call self%finalize()
418    end function
419
420    recursive subroutine operator_finalize(self)
421       class (type_base_operator), intent(inout) :: self
422       if (associated(self%previous)) then
423          if (self%previous%dereference()) deallocate(self%previous)
424       end if
425    end subroutine
426
427 end module output_manager_core
```