```
module output_manager

   use field_manager
   use output_manager_core
   use netcdf_output
   use text_output
   use output_operators_library
   use output_operators_time_average
   use output_operators_slice

   use yaml_settings

   implicit none

   private

   public output_manager_init, output_manager_start, output_manager_prepare_save, output_manager_save, output_manager_clean, &
      output_manager_add_file, global_attributes

   class (type_file), pointer :: first_file
   logical                    :: files_initialized
   logical, save, public, target :: allow_missing_fields = .false.
   type (type_logical_pointer), allocatable :: used(:)

   interface output_manager_save
      module procedure output_manager_save1
      module procedure output_manager_save2
   end interface

   type,extends(type_dictionary_populator) :: type_file_populator
      type (type_field_manager), pointer :: fm => null()
      character(len=:), allocatable :: title
      character(len=:), allocatable :: postfix
      logical :: ignore = .false.
   contains
      procedure :: create => process_file
   end type

   type,extends(type_list_populator) :: type_operator_populator
      type (type_field_manager), pointer             :: field_manager     => null()
      class (type_output_variable_settings), pointer :: variable_settings => null()
   contains
      procedure :: create => create_operator_settings
   end type

   type,extends(type_list_populator) :: type_group_populator
      class (type_file), pointer :: file
      class (type_output_variable_settings), pointer :: variable_settings => null()
   contains
      procedure :: create => create_group_settings
   end type

   type,extends(type_list_populator) :: type_variable_populator
      class (type_file), pointer :: file => null()
      class (type_output_variable_settings), pointer :: variable_settings => null()
   contains
      procedure :: create => create_variable_settings
   end type

   type (type_attributes) :: global_attributes

contains

   subroutine output_manager_init(field_manager, title, postfix, settings)
      type (type_field_manager), target :: field_manager
      character(len=*),           intent(in) :: title
      character(len=*), optional, intent(in) :: postfix
      class (type_settings), pointer, optional :: settings

      if (.not.associated(host)) then
         write (*,*) 'output_manager_init: the host of an output manager must set the host pointer before calling output_manager_init'
         stop 1
      end if
      nullify(first_file)
      files_initialized = .false.
      call configure_from_yaml(field_manager, title, postfix, settings)
   end subroutine

   subroutine output_manager_clean()
      class (type_file), pointer :: file, next

      file => first_file
      do while (associated(file))
         next => file%next
         call file%finalize()
         deallocate(file)
         file => next
      end do
      first_file => null()
      if (allocated(used)) deallocate(used)
   end subroutine

   subroutine populate(file)
      class (type_file), intent(inout) :: file

      type (type_output_item),               pointer :: item, next_item
```

```
 97       type (type_field_set)                          :: set
 98       class (type_field_set_member),         pointer :: member, next_member
 99       class (type_output_variable_settings), pointer :: output_settings
100       type (type_settings)                           :: settings
101       class (type_base_output_field),        pointer :: output_field, coordinate_field
102       type (type_dimension_pointer), allocatable     :: dimensions(:)
103       integer                                        :: i
104
105       ! First add fields selected by name
106       ! (they take priority over fields included with wildcard expressions)
107       item => file%first_item
108       do while (associated(item))
109          if (associated(item%field)) call create_field(item%settings, item%field, trim(item%name), .false.)
110          item => item%next
111       end do
112
113       item => file%first_item
114       do while (associated(item))
115          if (associated(item%category)) then
116             call host%log_message('Processing output category /'//trim(item%name)//':')
117             if (item%category%has_fields() .or. allow_missing_fields) then
118                ! Gather all variables in this category in a set
119                call item%category%get_all_fields(set, item%output_level)
120
121                ! Loop over all variables in the set and add them to the file
122                member => set%first
123                if (.not. associated(member)) call host%log_message('WARNING: output category "'//trim(item%name)//'" d
oes not contain any variables with requested output level.')
124                do while (associated(member))
125                   call host%log_message('  - '//trim(member%field%name))
126
127                   ! Create a separate object with variable settings, which gets initialized from the category settings
.
128                   ! The initialization routine is provide with a dummy (empty) configuration node, which gets cleaned
up right after.
129                   output_settings => file%create_settings()
130                   call output_settings%initialize(settings, item%settings)
131                   call settings%finalize()
132
133                   ! Add the variable
134                   call create_field(output_settings, member%field, trim(item%prefix) // trim(member%field%name) // tri
m(item%postfix), .true., item%category)
135
136                   ! Move to next set member and deallocate the current member (we are cleaning up the set as we go alo
ng)
137                   next_member => member%next
138                   deallocate(member)
139                   member => next_member
140                end do
141
142                ! Set is now empty (all items have been deallocated in previous loop)
143                set%first => null()
144
145                ! Deallocate the category settings (no longer needed; each contained variable has its own copy of the s
ettings)
146                call item%settings%finalize()
147                deallocate(item%settings)
148             else
149                call host%fatal_error('collect_from_categories','No variables have been registered under output categor
y "'//trim(item%name)//'".')
150             end if
151          end if
152
153          ! Move to next item and deallocate the current item (we are cleaning up the item list as we go along)
154          next_item => item%next
155          deallocate(item)
156          item => next_item
157       end do
158
159       ! Item list is now empty (all items have been deallocated in previous loop)
160       file%first_item => null()
161
162       output_field => file%first_field
163       do while (associated(output_field))
164          call output_field%get_metadata(dimensions=dimensions)
165          allocate(output_field%coordinates(size(dimensions)))
166          do i=1,size(dimensions)
167             if (.not.associated(dimensions(i)%p)) cycle
168             if (.not.associated(dimensions(i)%p%coordinate)) cycle
169             coordinate_field => find_field(dimensions(i)%p%coordinate%name)
170             if (.not. associated(coordinate_field)) then
171                coordinate_field => output_field%get_field(dimensions(i)%p%coordinate)
172                if (associated(coordinate_field)) call append_field(trim(dimensions(i)%p%coordinate%name), coordinate_f
ield, file%create_settings())
173             end if
174             if (associated(coordinate_field)) coordinate_field%is_coordinate = .true.
175             output_field%coordinates(i)%p => coordinate_field
176          end do
177          output_field => output_field%next
178       end do
179
180    contains
181
182       function find_field(output_name) result(field)
183          character(len=*), intent(in)           :: output_name
184          class (type_base_output_field), pointer :: field
185          field => file%first_field
186          do while (associated(field))
```

```
187            if (field%output_name == output_name) return
188            field => field%next
189         end do
190      end function
191
192      subroutine create_field(output_settings, field, output_name, ignore_if_exists, category)
193         class (type_output_variable_settings), target :: output_settings
194         type (type_field),                     target :: field
195         character(len=*), intent(in)                  :: output_name
196         logical,          intent(in)                  :: ignore_if_exists
197         class (type_category_node), optional,  target :: category
198
199         class (type_base_output_field), pointer :: output_field
200
201         output_field => find_field(output_name)
202         if (associated(output_field)) then
203            if (.not. ignore_if_exists) call host%fatal_error('create_field', 'A different output field with name "'//
    output_name//'" already exists.')
204            return
205         end if
206
207         output_field => wrap_field(field, allow_missing_fields)
208
209         if (associated(output_settings%final_operator)) output_field => output_settings%final_operator%apply_all(outp
    ut_field)
210         if (associated(output_field)) call append_field(output_name, output_field, output_settings, category)
211      end subroutine
212
213      subroutine append_field(output_name, output_field, output_settings, category)
214         character(len=*),               intent(in)          :: output_name
215         class (type_base_output_field), intent(inout), target :: output_field
216         class (type_output_variable_settings),         target :: output_settings
217         class (type_category_node), optional,          target :: category
218
219         class (type_base_output_field), pointer :: last_field
220
221         output_field%settings => output_settings
222         output_field%output_name = trim(output_name)
223         if (present(category)) output_field%category => category
224
225         if (associated(file%first_field)) then
226            last_field => file%first_field
227            do while (associated(last_field%next))
228               last_field => last_field%next
229            end do
230            last_field%next => output_field
231         else
232            file%first_field => output_field
233         end if
234      end subroutine
235
236   end subroutine populate
237
238   subroutine output_manager_start(julianday,secondsofday,microseconds,n)
239      integer, intent(in) :: julianday,secondsofday,microseconds,n
240
241      class (type_file), pointer :: file
242
243      file => first_file
244      do while (associated(file))
245         call populate(file)
246
247         ! If we do not have a start time yet, use current.
248         if (file%first_julian <= 0) then
249            file%first_julian = julianday
250            file%first_seconds = secondsofday
251         end if
252
253         ! Create output file
254         call file%initialize()
255
256         file => file%next
257      end do
258      if (associated(first_file)) then
259         call first_file%field_manager%collect_used(used)
260      else
261         allocate(used(0))
262      end if
263      files_initialized = .true.
264   end subroutine
265
266   subroutine set_next_output(self, julianday, secondsofday, microseconds, n)
267      class (type_file), intent(inout) :: self
268      integer,           intent(in)    :: julianday, secondsofday, microseconds, n
269
270      integer                              :: yyyy,mm,dd,yyyy0,mm0
271      integer(kind=selected_int_kind(12)) :: offset
272
273      if (self%time_unit == time_unit_none) return
274
275      ! Determine time (julian day, seconds of day) for first output.
276      self%next_julian = self%first_julian
277      self%next_seconds = self%first_seconds
278      offset = 86400*(julianday-self%first_julian) + (secondsofday-self%first_seconds)
279      if (offset > 0) then
280         select case (self%time_unit)
281            case (time_unit_second)
282               self%next_seconds = self%next_seconds + ((offset+self%time_step-1)/self%time_step)*self%time_step
```

```
283                self%next_julian = self%next_julian + self%next_seconds/86400
284                self%next_seconds = mod(self%next_seconds,86400)
285             case (time_unit_hour)
286                self%next_seconds = self%next_seconds + ((offset+self%time_step*3600-1)/(self%time_step*3600))*self%tim
    e_step*3600
287                self%next_julian = self%next_julian + self%next_seconds/86400
288                self%next_seconds = mod(self%next_seconds,86400)
289             case (time_unit_day)
290                self%next_julian = self%next_julian + ((offset+self%time_step*86400-1)/(self%time_step*86400))*self%tim
    e_step
291             case (time_unit_month)
292                call host%calendar_date(julianday,yyyy,mm,dd)
293                call host%calendar_date(self%first_julian,yyyy,mm0,dd)
294                mm = mm0 + ((mm-mm0+self%time_step-1)/self%time_step)*self%time_step
295                yyyy = yyyy + (mm-1)/12
296                mm = mod(mm-1,12)+1
297                call host%julian_day(yyyy,mm,dd,self%next_julian)
298                if (self%next_julian == julianday .and. secondsofday > self%next_seconds) then
299                   mm = mm + self%time_step
300                   yyyy = yyyy + (mm-1)/12
301                   mm = mod(mm-1,12)+1
302                   call host%julian_day(yyyy,mm,dd,self%next_julian)
303                end if
304             case (time_unit_year)
305                call host%calendar_date(julianday,yyyy,mm,dd)
306                call host%calendar_date(self%first_julian,yyyy0,mm,dd)
307                yyyy = yyyy0 + ((yyyy-yyyy0+self%time_step-1)/self%time_step)*self%time_step
308                call host%julian_day(yyyy,mm,dd,self%next_julian)
309                if (self%next_julian == julianday .and. secondsofday > self%next_seconds) then
310                   yyyy = yyyy + self%time_step
311                   call host%julian_day(yyyy,mm,dd,self%next_julian)
312                end if
313          end select
314       end if
315    end subroutine
316
317    subroutine output_manager_save1(julianday,secondsofday,n)
318       integer,intent(in) :: julianday,secondsofday,n
319       call output_manager_save2(julianday,secondsofday,0,n)
320    end subroutine
321
322    subroutine output_manager_prepare_save(julianday, secondsofday, microseconds, n)
323       integer,intent(in) :: julianday, secondsofday, microseconds, n
324
325       integer                                  :: i
326       class (type_file),              pointer :: file
327       class (type_base_output_field), pointer :: output_field
328       logical                                  :: required
329
330       if (.not. files_initialized) call output_manager_start(julianday, secondsofday, microseconds, n)
331
332       ! Start by marking all fields as not needing computation
333       do i = 1, size(used)
334          used(i)%p = .false.
335       end do
336
337       file => first_file
338       do while (associated(file))
339          if (in_window(file, julianday, secondsofday, microseconds, n)) then
340             if (file%next_julian == -1) call set_next_output(file, julianday, secondsofday, microseconds, n)
341             output_field => file%first_field
342             do while (associated(output_field))
343                select case (file%time_unit)
344                case (time_unit_dt)
345                   required = file%first_index == -1 .or. mod(n - file%first_index, file%time_step) == 0
346                case default
347                   required = file%next_julian == -1 .or. (julianday == file%next_julian .and. secondsofday >= file%nex
    t_seconds) .or. julianday > file%next_julian
348                end select
349                call output_field%flag_as_required(required)
350                output_field => output_field%next
351             end do
352          end if
353          file => file%next
354       end do
355    end subroutine
356
357    logical function in_window(self, julianday, secondsofday, microseconds, n)
358       class (type_file), intent(in) :: self
359       integer,           intent(in) :: julianday, secondsofday, microseconds, n
360
361       in_window = ((julianday == self%first_julian .and. secondsofday >= self%first_seconds) .or. julianday > self%fir
    st_julian) &
362             .and. ((julianday == self%last_julian .and. secondsofday <= self%last_seconds)  .or. julianday < self%last
    _julian)
363    end function
364
365    subroutine output_manager_save2(julianday,secondsofday,microseconds,n)
366       integer,intent(in) :: julianday,secondsofday,microseconds,n
367
368       class (type_file),              pointer :: file
369       class (type_base_output_field), pointer :: output_field
370       integer                                  :: yyyy,mm,dd
371       logical                                  :: output_required
372
373       if (.not.files_initialized) call output_manager_start(julianday,secondsofday,microseconds,n)
374
375       file => first_file
```

```
376       do while (associated(file))
377          if (in_window(file, julianday, secondsofday, microseconds, n)) then
378             if (file%next_julian == -1) call set_next_output(file, julianday, secondsofday, microseconds, n)
379
380             ! Increment time-integrated fields
381             output_field => file%first_field
382             do while (associated(output_field))
383                call output_field%new_data()
384                output_field => output_field%next
385             end do
386
387             ! Determine whether output is required
388             select case (file%time_unit)
389             case (time_unit_none)
390                output_required = .false.
391             case (time_unit_dt)
392                if (file%first_index == -1) file%first_index = n
393                output_required = mod(n - file%first_index, file%time_step) == 0
394             case default
395                output_required  = (julianday == file%next_julian .and. secondsofday >= file%next_seconds) .or. juliand
    ay > file%next_julian
396             end select
397
398             if (output_required) then
399                output_field => file%first_field
400                do while (associated(output_field))
401                   call output_field%before_save()
402                   output_field => output_field%next
403                end do
404
405                ! Do output
406                call file%save(julianday,secondsofday,microseconds)
407
408                ! Determine time (julian day, seconds of day) for next output.
409                select case (file%time_unit)
410                case (time_unit_second)
411                   file%next_seconds = file%next_seconds + file%time_step
412                   file%next_julian = file%next_julian + file%next_seconds/86400
413                   file%next_seconds = mod(file%next_seconds,86400)
414                case (time_unit_hour)
415                   file%next_seconds = file%next_seconds + file%time_step*3600
416                   file%next_julian = file%next_julian + file%next_seconds/86400
417                   file%next_seconds = mod(file%next_seconds,86400)
418                case (time_unit_day)
419                   file%next_julian = file%next_julian + file%time_step
420                case (time_unit_month)
421                   call host%calendar_date(julianday,yyyy,mm,dd)
422                   mm = mm + file%time_step
423                   yyyy = yyyy + (mm-1)/12
424                   mm = mod(mm-1,12)+1
425                   call host%julian_day(yyyy,mm,dd,file%next_julian)
426                case (time_unit_year)
427                   call host%calendar_date(julianday,yyyy,mm,dd)
428                   yyyy = yyyy + file%time_step
429                   call host%julian_day(yyyy,mm,dd,file%next_julian)
430                end select
431             end if
432
433          end if ! in output time window
434
435          file => file%next
436       end do
437    end subroutine output_manager_save2
438
439    subroutine configure_from_yaml(field_manager, title, postfix, settings)
440       type (type_field_manager), target      :: field_manager
441       character(len=*),           intent(in) :: title
442       character(len=*), optional, intent(in) :: postfix
443       class (type_settings), pointer, optional :: settings
444
445       logical                         :: file_exists
446       integer,parameter               :: yaml_unit = 100
447       class (type_settings), pointer :: settings_
448       type (type_settings)           :: ignored_settings
449       type (type_file_populator)     :: populator
450
451       inquire(file='output.yaml',exist=file_exists)
452
453       populator%fm => field_manager
454       populator%title = trim(title)
455       if (present(postfix)) populator%postfix = postfix
456
457       if (present(settings)) then
458          settings_ => settings
459       else
460          allocate(settings_)
461       end if
462
463       if (file_exists) then
464          ! Settings from output.yaml
465          if (present(settings)) then
466             ! yaml node also provided, but output.yaml takes priority.
467             ! Read yaml node values into a dummy settings object, so everything is still properly parsed and the user
    receives warnings.
468             populator%ignore = .true.
469             ignored_settings%backing_store_node => settings%backing_store_node
470             call ignored_settings%populate(populator)
471             populator%ignore = .false.
```

```
472          end if
473          call settings_%load('output.yaml', yaml_unit)
474       elseif (.not. present(settings)) then
475          call host%log_message('WARNING: no output files will be written because output settings have not been provide
     d.')
476          return
477       end if
478       call settings_%populate(populator)
479       if (file_exists) then
480          if (.not. settings_%check_all_used()) call host%fatal_error('configure_from_yaml', 'Unknown setting(s) in out
     put.yaml.')
481       end if
482       if (.not. present(settings)) then
483          call settings_%finalize()
484          deallocate(settings_)
485       end if
486    end subroutine configure_from_yaml
487
488    subroutine output_manager_add_file(field_manager, file)
489       type (type_field_manager), target :: field_manager
490       class (type_file),         target :: file
491
492       file%field_manager => field_manager
493       file%next => first_file
494       first_file => file
495    end subroutine output_manager_add_file
496
497    subroutine process_file(self, pair)
498       class (type_file_populator), intent(inout) :: self
499       type (type_key_value_pair),  intent(inout) :: pair
500
501       class (type_logical_setting) ,pointer :: logical_setting
502       integer                               :: fmt
503       class (type_file), pointer            :: file
504       character(len=:), allocatable         :: string
505       class (type_settings),       pointer :: file_settings
506       logical :: success
507       class (type_output_variable_settings), pointer :: variable_settings
508
509       type (type_dimension),       pointer :: dim
510       integer                               :: global_start, global_stop, stride
511       logical                               :: is_active
512       class (type_slice_operator), pointer :: slice_operator
513       integer                               :: display
514
515       if (pair%key == 'allow_missing_fields') then
516          logical_setting => type_logical_setting_create(pair, 'ignore unknown requested output fields', target=allow_m
     issing_fields)
517          return
518       end if
519
520       file_settings => type_settings_create(pair, 'path of output file, excluding extension')
521
522       is_active = file_settings%get_logical('is_active', 'write output to this file', default=.true., display=display_
     hidden)
523       is_active = file_settings%get_logical('use', 'write output to this file', default=.true., display=display_advanc
     ed)
524       if (self%ignore) then
525          call host%log_message('WARNING: '//pair%value%path//' will be ignored because output.yaml is present.')
526          is_active = .false.
527       end if
528 #ifdef NETCDF_FMT
529       fmt = file_settings%get_integer('format', 'format', options=(/option(1, 'text', 'text'), option(2, 'NetCDF', 'ne
     tcdf')/), default=2, display=display_advanced)
530 #else
531       fmt = file_settings%get_integer('format', 'format', options=(/option(1, 'text', 'text')/), default=1, display=di
     splay_advanced)
532 #endif
533
534       select case (fmt)
535       case (1)
536          allocate(type_text_file::file)
537       case (2)
538 #ifdef NETCDF_FMT
539          allocate(type_netcdf_file::file)
540 #endif
541       end select
542       call file%attributes%update(global_attributes)
543
544       ! Create file object and prepend to list.
545       file%path = pair%name
546       if (allocated(self%postfix)) file%postfix = self%postfix
547       file%field_manager => self%fm
548       if (is_active) call output_manager_add_file(self%fm, file)
549
550       ! Can be used for CF global attributes
551       call file_settings%get(file%title, 'title', 'title', default=self%title, display=display_advanced)
552       call file_settings%get(file%time_unit, 'time_unit', 'time unit', default=time_unit_day, options=(/ &
553          option(time_unit_second, 'second', 'second'), option(time_unit_hour, 'hour', 'hour'), option(time_unit_day, '
     day', 'day'), &
554          option(time_unit_month, 'month', 'month'), option(time_unit_year, 'year', 'year'), option(time_unit_dt, 'mode
     l time step', 'dt')/))
555
556       ! Determine time step
557       call file_settings%get(file%time_step, 'time_step', 'number of time units between output', minimum=1, default=1)

558       string = file_settings%get_string('time_start', 'start', 'yyyy-mm-dd HH:MM:SS', default='', display=display_adva
     nced)
```

```
559       if (string /= '') then
560          call read_time_string(string, file%first_julian, file%first_seconds, success)
561          if (.not. success) call host%fatal_error('process_file','Error in output configuration: invalid time_start "'
    //string//'" specified for file "'//pair%name//'". Required format: yyyy-mm-dd HH:MM:SS.')
562       end if
563       string = file_settings%get_string('time_stop', 'stop', 'yyyy-mm-dd HH:MM:SS', default='', display=display_advanc
    ed)
564       if (string /= '') then
565          call read_time_string(string, file%last_julian, file%last_seconds, success)
566          if (.not. success) call host%fatal_error('process_file','Error in output configuration: invalid time_stop "'/
    /string//'" specified for file "'//pair%name//'". Required format: yyyy-mm-dd HH:MM:SS.')
567       end if
568
569       ! Determine dimension ranges
570       allocate(slice_operator)
571       dim => self%fm%first_dimension
572       do while (associated(dim))
573          if (dim%iterator /= '') then
574             display = display_advanced
575             if (dim%global_length == 1) display = display_hidden
576             global_start = file_settings%get_integer(trim(dim%iterator)//'_start', 'start index for '//trim(dim%iterat
    or)//' dimension', default=1, minimum=0, maximum=dim%global_length, display=display)
577             global_stop = file_settings%get_integer(trim(dim%iterator)//'_stop', 'stop index for '//trim(dim%iterator)
    //' dimension', default=dim%global_length, minimum=1, maximum=dim%global_length, display=display)
578             if (global_start > global_stop) call host%fatal_error('process_file','Error parsing output.yaml: '//trim(d
    im%iterator)//'_stop must equal or exceed '//trim(dim%iterator)//'_start')
579             stride = file_settings%get_integer(trim(dim%iterator)//'_stride', 'stride for '//trim(dim%iterator)//' dim
    ension', default=1, minimum=1, display=display)
580             call slice_operator%add(trim(dim%name), global_start, global_stop, stride)
581          end if
582          dim => dim%next
583       end do
584       variable_settings => file%create_settings()
585       call variable_settings%initialize(file_settings)
586       variable_settings%final_operator => slice_operator
587
588       ! Allow specific file implementation to parse additional settings from yaml file.
589       call file%configure(file_settings)
590
591       call configure_group(file, file_settings, variable_settings)
592       call variable_settings%finalize()
593       deallocate(variable_settings)
594    end subroutine process_file
595
596    recursive subroutine configure_group(file, settings, variable_settings)
597       class (type_file), target, intent(inout) :: file
598       class (type_settings),     intent(inout) :: settings
599       class (type_output_variable_settings), target :: variable_settings
600
601       type (type_operator_populator) :: operator_populator
602       type (type_group_populator)    :: group_populator
603       type (type_variable_populator) :: variable_populator
604
605       ! Get operators
606       operator_populator%field_manager => file%field_manager
607       operator_populator%variable_settings => variable_settings
608       call settings%get_list('operators', operator_populator, display=display_advanced)
609
610       ! Get list with groups [if any]
611       group_populator%file => file
612       group_populator%variable_settings => variable_settings
613       call settings%get_list('groups', group_populator, display=display_advanced)
614
615       ! Get list with variables
616       variable_populator%file => file
617       variable_populator%variable_settings => variable_settings
618       call settings%get_list('variables', variable_populator)
619    end subroutine
620
621    recursive subroutine create_group_settings(self, index, item)
622       class (type_group_populator), intent(inout) :: self
623       integer,                      intent(in)    :: index
624       type (type_list_item),        intent(inout) :: item
625
626       class (type_settings),                 pointer :: group_settings
627       class (type_output_variable_settings), pointer :: variable_settings
628
629       ! Obtain dictionary with user-provided settings
630       group_settings => type_settings_create(item)
631
632       ! Create object that will contain final output settings for this group
633       variable_settings => self%file%create_settings()
634       call variable_settings%initialize(group_settings, self%variable_settings)
635
636       ! Configure output settings
637       ! This will also read additional user options from variable_settings and apply them to group_settings.
638       call configure_group(self%file, group_settings, variable_settings)
639       call variable_settings%finalize()
640       deallocate(variable_settings)
641    end subroutine
642
643    recursive subroutine create_operator_settings(self, index, item)
644       class (type_operator_populator), intent(inout) :: self
645       integer,                         intent(in)    :: index
646       type (type_list_item),           intent(inout) :: item
647
648       class (type_settings), pointer :: settings
649
```

```
650       settings => type_settings_create(item)
651       call apply_operator(self%variable_settings%final_operator, settings, self%field_manager)
652    end subroutine
653
654    recursive subroutine create_variable_settings(self, index, item)
655       class (type_variable_populator), intent(inout) :: self
656       integer,                         intent(in)    :: index
657       type (type_list_item),           intent(inout) :: item
658
659       class (type_settings),              pointer :: variable_settings
660       character(len=:), allocatable               :: source_name
661       type (type_output_item),            pointer :: output_item
662       class (type_time_average_operator), pointer :: time_filter
663       integer                                     :: n
664
665       variable_settings => type_settings_create(item)
666
667       ! Name of source variable
668       source_name = variable_settings%get_string('source', 'variable name in model')
669
670       allocate(output_item)
671       output_item%settings => self%file%create_settings()
672       call output_item%settings%initialize(variable_settings, self%variable_settings)
673
674       ! If this output is not instantaneous, add time averaging/integration to the operator stack
675       if (output_item%settings%time_method /= time_method_instantaneous .and. output_item%settings%time_method /= time
    _method_none) then
676          allocate(time_filter)
677          time_filter%method = output_item%settings%time_method
678          time_filter%previous => output_item%settings%final_operator
679          output_item%settings%final_operator => time_filter
680       end if
681
682       ! Ðetermine whether to create an output field or an output category
683       n = len(source_name)
684       if (source_name(n:n)=='*') then
685          if (n==1) then
686             output_item%name = ''
687          else
688             output_item%name = source_name(:n-2)
689          end if
690
691          ! Prefix for output name
692          call variable_settings%get(output_item%prefix, 'prefix', 'name prefix used in output', default='', display=di
    splay_advanced)
693
694          ! Postfix for output name
695          call variable_settings%get(output_item%postfix, 'postfix', 'name postfix used in output', default='', display
    =display_advanced)
696
697          ! Output level
698          call variable_settings%get(output_item%output_level, 'output_level', 'output level', default=output_level_def
    ault, display=display_advanced)
699       else
700          output_item%field => self%file%field_manager%select_for_output(source_name)
701
702          ! Name of output variable (may differ from source name)
703          call variable_settings%get(output_item%name, 'name', 'name used in output', default=source_name, display=disp
    lay_advanced)
704       end if
705       call self%file%append_item(output_item)
706    end subroutine
707
708 end module
```