```
#include "fabm_driver.h"
#include "fabm_private.h"

! This module define standard expressions that can be used by biogeochemical models.

module fabm_expressions

   use fabm_types
   use fabm_driver

   implicit none

   private

   public temporal_mean, temporal_maximum, vertical_mean, vertical_integral
   public type_interior_temporal_mean, type_horizontal_temporal_mean, type_horizontal_temporal_maximum, type_vertical_integral

   type, extends(type_interior_expression) :: type_interior_temporal_mean
      real(rk) :: period   ! Time period to average over (s)
      integer  :: n
      real(rk) :: missing_value = -2.e20_rk
      type (type_link), pointer :: link => null()
      integer :: in = -1

      real(rk), private :: previous_time, bin_end_time
      integer,  private :: ioldest  = -1
      integer,  private :: icurrent = -1
      logical,  private :: complete = .false.
      real(rke), allocatable _DIMENSION_GLOBAL_PLUS_1_ :: history
#if _FABM_DIMENSION_COUNT_>0
      real(rke), allocatable _DIMENSION_GLOBAL_ :: previous_value, last_exact_mean, mean
#else
      real(rke) :: previous_value, last_exact_mean, mean
#endif
   contains
      procedure :: update => interior_temporal_mean_update
   end type

   type, extends(type_horizontal_expression) :: type_horizontal_temporal_mean
      real(rk) :: period   ! Time period to average over (s)
      integer  :: n
      real(rk) :: missing_value = -2.e20_rk
      real(rk) :: last_time, next_save_time

      type (type_link), pointer :: link => null()
      integer :: in = -1

      real(rk), private :: previous_time, bin_end_time
      integer :: ioldest  = -1
      integer :: icurrent = -1
      logical,  private :: complete = .false.
      real(rke), allocatable _DIMENSION_GLOBAL_HORIZONTAL_PLUS_1_ :: history
#if _HORIZONTAL_DIMENSION_COUNT_>0
      real(rke), allocatable _DIMENSION_GLOBAL_HORIZONTAL_ :: previous_value, last_exact_mean, mean
#else
      real(rke) :: previous_value, last_exact_mean, mean
#endif
   end type

   type, extends(type_horizontal_expression) :: type_horizontal_temporal_maximum
      real(rk) :: period                    ! Time window to compute running maximum over (s)
      integer  :: n                         ! Number of bins to use to cover the period
      real(rk) :: missing_value = -2.e20_rk  ! Missing value to use until the simulation has covered the window size [period]

      type (type_link), pointer :: link => null()
      integer :: in = -1

      real(rk), private :: previous_time, bin_end_time
      integer :: icurrent = -1
      logical,  private :: complete = .false.
      real(rke), allocatable _DIMENSION_GLOBAL_HORIZONTAL_PLUS_1_ :: history
#if _HORIZONTAL_DIMENSION_COUNT_>0
      real(rke), allocatable _DIMENSION_GLOBAL_HORIZONTAL_ :: previous_value, maximum
#else
      real(rke) :: previous_value, maximum
#endif
   contains
      procedure :: update => horizontal_temporal_maximum_update
   end type

   type, extends(type_horizontal_expression) :: type_vertical_integral
      real(rk) :: minimum_depth = 0.0_rk        ! Depth below surface in m (positive)
      real(rk) :: maximum_depth = huge(1.0_rk)  ! Depth below surface in m (positive)
      logical  :: average       = .false.       ! Whether to divide the depth integral by water depth, thus computing the vertical average
      character(len=attribute_length) :: input_name = ''

      type (type_link), pointer :: link => null()
   end type

   interface temporal_mean
      module procedure interior_temporal_mean
      module procedure horizontal_temporal_mean
   end interface

   interface temporal_maximum
```

```
 96       module procedure horizontal_temporal_maximum
 97    end interface
 98
 99    interface vertical_mean
100       module procedure vertical_dependency_mean
101       module procedure vertical_state_mean
102    end interface
103
104    interface vertical_integral
105       module procedure vertical_dependency_integral
106       module procedure vertical_state_integral
107    end interface
108
109 contains
110
111    function vertical_dependency_mean(input, minimum_depth, maximum_depth) result(expression)
112       type (type_dependency_id), intent(inout), target :: input
113       real(rk), optional,        intent(in)            :: minimum_depth,maximum_depth
114       type (type_vertical_integral)                    :: expression
115       expression = vertical_integral(input,minimum_depth,maximum_depth,average=.true.)
116    end function
117
118    function vertical_state_mean(input, minimum_depth, maximum_depth) result(expression)
119       type (type_state_variable_id), intent(inout), target :: input
120       real(rk), optional,            intent(in)            :: minimum_depth, maximum_depth
121       type (type_vertical_integral)                        :: expression
122       expression = vertical_integral_generic(input,minimum_depth,maximum_depth,average=.true.)
123    end function
124
125    function vertical_dependency_integral(input, minimum_depth, maximum_depth, average) result(expression)
126       type (type_dependency_id), intent(inout),target :: input
127       real(rk), optional,        intent(in)           :: minimum_depth, maximum_depth
128       logical,  optional,        intent(in)           :: average
129       type (type_vertical_integral)                     :: expression
130       expression = vertical_integral_generic(input, minimum_depth, maximum_depth, average)
131    end function
132
133    function vertical_state_integral(input, minimum_depth, maximum_depth, average) result(expression)
134       type (type_state_variable_id), intent(inout), target :: input
135       real(rk), optional,            intent(in)            :: minimum_depth, maximum_depth
136       logical,  optional,            intent(in)            :: average
137       type (type_vertical_integral)                          :: expression
138       expression = vertical_integral_generic(input, minimum_depth, maximum_depth, average)
139    end function
140
141    function vertical_integral_generic(input, minimum_depth, maximum_depth, average) result(expression)
142       class (type_variable_id), intent(inout), target :: input
143       real(rk), optional,       intent(in)            :: minimum_depth, maximum_depth
144       logical,  optional,       intent(in)            :: average
145       type (type_vertical_integral)                   :: expression
146
147       character(len=attribute_length) :: postfix
148
149       if (.not. associated(input%link)) call fatal_error('fabm_expressions::vertical_mean', &
150          'Input variable has not been registered yet.')
151       expression%input_name = input%link%target%name
152
153       ! Create a name for the expression
154       postfix = ''
155       if (present(minimum_depth) .and. present(maximum_depth)) then
156          if (minimum_depth > maximum_depth) call fatal_error('fabm_expressions::vertical_mean', &
157             'Minimum depth exceeds maximum depth.')
158          write (postfix,'(a,i0,a,i0,a)') '_between_', int(minimum_depth), '_m_and_', int(maximum_depth), '_m'
159       elseif (present(minimum_depth)) then
160          write (postfix,'(a,i0,a)') '_below_', int(minimum_depth), '_m'
161       elseif (present(maximum_depth)) then
162          write (postfix,'(a,i0,a)') '_above_', int(maximum_depth), '_m'
163       end if
164       if (present(average)) expression%average = average
165
166       if (expression%average) then
167          expression%output_name = 'vertical_mean_' // trim(input%link%name) // trim(postfix)
168       else
169          expression%output_name = 'integral_of_' // trim(input%link%name) // '_wrt_depth' // trim(postfix)
170       end if
171
172       expression%link => input%link
173       if (present(minimum_depth)) expression%minimum_depth = minimum_depth
174       if (present(maximum_depth)) expression%maximum_depth = maximum_depth
175    end function
176
177    function interior_temporal_mean(input, period, resolution, missing_value) result(expression)
178       type (type_dependency_id), intent(inout), target :: input
179       real(rk),                  intent(in)            :: period, resolution
180       real(rk), optional,        intent(in)            :: missing_value
181       type (type_interior_temporal_mean)               :: expression
182
183       character(len=attribute_length) :: prefix, postfix
184
185       if (.not. associated(input%link)) call fatal_error('fabm_expressions::interior_temporal_mean', &
186          'Input variable has not been registered yet.')
187
188       write (expression%output_name,'(i0,a,a,a,i0,a)') int(period), '_s_mean_', trim(input%link%name), '_at_', int(res
    olution), '_s_resolution'
189       expression%link => input%link
190       expression%n = nint(period / resolution)
191       expression%period = period
192       if (present(missing_value)) expression%missing_value = missing_value
```

```
193    end function
194
195    function horizontal_temporal_mean(input, period, resolution, missing_value) result(expression)
196       class (type_horizontal_dependency_id), intent(inout), target :: input
197       real(rk),                              intent(in)            :: period, resolution
198       real(rk), optional,                    intent(in)            :: missing_value
199       type (type_horizontal_temporal_mean)                         :: expression
200
201       character(len=attribute_length) :: prefix, postfix
202
203       if (.not. associated(input%link)) call fatal_error('fabm_expressions::horizontal_temporal_mean', &
204          'Input variable has not been registered yet.')
205
206       write (expression%output_name,'(i0,a,a,a,i0,a)') int(period), '_s_mean_', trim(input%link%name), '_at_', int(res
olution), '_s_resolution'
207       expression%link => input%link
208       expression%n = nint(period / resolution)
209       expression%period = period
210       if (present(missing_value)) expression%missing_value = missing_value
211    end function
212
213    function horizontal_temporal_maximum(input, period, resolution, missing_value) result(expression)
214       class (type_horizontal_dependency_id), intent(inout), target :: input
215       real(rk),                              intent(in)            :: period, resolution
216       real(rk), optional,                    intent(in)            :: missing_value
217       type (type_horizontal_temporal_maximum)                      :: expression
218
219       if (.not. associated(input%link)) call fatal_error('fabm_expressions::horizontal_temporal_max', &
220          'Input variable has not been registered yet.')
221
222       write (expression%output_name,'(i0,a,a,a,i0,a)') int(period), '_s_max_', trim(input%link%name), '_at_', int(reso
lution), '_s_resolution'
223       expression%link => input%link
224       expression%n = nint(period / resolution)
225       expression%period = period
226       if (present(missing_value)) expression%missing_value = missing_value
227    end function
228
229    subroutine interior_temporal_mean_update(self, time, value _POSTARG_LOCATION_RANGE_)
230       class (type_interior_temporal_mean), intent(inout) :: self
231       real(rke),                           intent(in)    :: time
232       real(rke) _ATTRIBUTES_GLOBAL_,       intent(in)    :: value
233       _DECLARE_ARGUMENTS_LOCATION_RANGE_
234
235       integer  :: ibin
236       real(rke) :: dt, w, dt_bin
237       _DECLARE_LOCATION_
238
239       ! Note that all array processing below uses explicit loops in order to respect
240       ! any limits on the active domain given by the _LOCATION_RANGE_ argument.
241
242       dt_bin = self%period / self%n
243
244       if (self%ioldest == -1) then
245          ! Start of simulation
246          self%previous_time = time
247          self%bin_end_time = time + dt_bin
248          self%icurrent = 1
249          self%ioldest = 2
250          self%previous_value = 0.0_rke
251       end if
252
253       do while (time >= self%bin_end_time)
254          ! Linearly interpolate to value at end-of-bin time and add that to the current bin
255          dt = self%bin_end_time - self%previous_time
256          w = dt / (time - self%previous_time)   ! weight for current time (leaving 1-w for previous time)
257          _BEGIN_GLOBAL_LOOP_
258             self%history(_PREARG_LOCATION_ self%icurrent) = self%history(_PREARG_LOCATION_ self%icurrent) &
259                + ((1._rke - 0.5_rke * w) * self%previous_value _INDEX_LOCATION_ + 0.5_rke * w * value _INDEX_LOCATION_
) &
260                * dt / self%period
261          _END_GLOBAL_LOOP_
262
263          if (self%complete) then
264             ! We already had a complete history (bins covering the full window size). Add the newly full bin, subtract
 the oldest bin
265             _BEGIN_GLOBAL_LOOP_
266                self%last_exact_mean _INDEX_LOCATION_ = self%last_exact_mean _INDEX_LOCATION_ &
267                   - self%history(_PREARG_LOCATION_ self%ioldest) + self%history(_PREARG_LOCATION_ self%icurrent)
268             _END_GLOBAL_LOOP_
269          elseif (self%icurrent == self%n) then
270             ! We just completed our history. Create the mean by summing all filled bins.
271             do ibin = 1, self%n
272                _BEGIN_GLOBAL_LOOP_
273                   self%last_exact_mean _INDEX_LOCATION_ = self%last_exact_mean _INDEX_LOCATION_  &
274                      + self%history(_PREARG_LOCATION_ ibin)
275                _END_GLOBAL_LOOP_
276             end do
277             self%complete = .true.
278          end if
279
280          ! Update previous time and value to match end of current bin
281          self%previous_time = self%bin_end_time
282          _BEGIN_GLOBAL_LOOP_
283             self%previous_value _INDEX_LOCATION_ = (1._rke - w) * self%previous_value _INDEX_LOCATION_ + w * value _IN
DEX_LOCATION_
284          _END_GLOBAL_LOOP_
285
```

```
286          ! Move to next bin: update indices, end time and empty newly current bin
287          self%icurrent = self%ioldest
288          self%ioldest = self%ioldest + 1
289          if (self%ioldest > self%n + 1) self%ioldest = 1
290          self%bin_end_time = self%bin_end_time + dt_bin
291          _BEGIN_GLOBAL_LOOP_
292             self%history(_PREARG_LOCATION_ self%icurrent) = 0
293          _END_GLOBAL_LOOP_
294       end do
295
296       ! Compute average of previous and current value, multiply by time difference, pre-divide by window size, and add to current bin.
297       _BEGIN_GLOBAL_LOOP_
298          self%history(_PREARG_LOCATION_ self%icurrent) = self%history(_PREARG_LOCATION_ self%icurrent) &
299             + 0.5_rke * (self%previous_value _INDEX_LOCATION_ + value _INDEX_LOCATION_) &
300             * (time - self%previous_time) / self%period
301       _END_GLOBAL_LOOP_
302
303       if (self%complete) then
304          ! We have a full history covering at least one window size. Update the running mean.
305          ! The result is an approximation that assumes linear change over the period covered by the oldest bin.
306          _BEGIN_GLOBAL_LOOP_
307             self%mean _INDEX_LOCATION_ = self%last_exact_mean _INDEX_LOCATION_ &
308                + self%history(_PREARG_LOCATION_ self%icurrent) &
309                - self%history(_PREARG_LOCATION_ self%ioldest) * (time - self%bin_end_time + dt_bin) / dt_bin
310          _END_GLOBAL_LOOP_
311       end if
312
313       ! Store current time and value to enable linear interpolation in subsequent call.
314       self%previous_time = time
315       _BEGIN_GLOBAL_LOOP_
316          self%previous_value _INDEX_LOCATION_ = value _INDEX_LOCATION_
317       _END_GLOBAL_LOOP_
318    end subroutine
319
320    subroutine horizontal_temporal_mean_update(self, time, value _POSTARG_HORIZONTAL_LOCATION_RANGE_)
321       class (type_horizontal_temporal_mean),    intent(inout) :: self
322       real(rke),                                intent(in)    :: time
323       real(rke) _ATTRIBUTES_GLOBAL_HORIZONTAL_, intent(in)    :: value
324       _DECLARE_ARGUMENTS_HORIZONTAL_LOCATION_RANGE_
325
326       integer  :: ibin
327       real(rke) :: dt, w, dt_bin
328       _DECLARE_HORIZONTAL_LOCATION_
329
330       ! Note that all array processing below uses explicit loops in order to respect
331       ! any limits on the active domain given by the _HORIZONTAL_LOCATION_RANGE_ argument.
332
333       dt_bin = self%period / self%n
334
335       if (self%ioldest == -1) then
336          ! Start of simulation
337          self%previous_time = time
338          self%bin_end_time = time + dt_bin
339          self%icurrent = 1
340          self%ioldest = 2
341          self%previous_value = 0.0_rke
342       end if
343
344       do while (time >= self%bin_end_time)
345          ! Linearly interpolate to value at end-of-bin time and add that to the current bin
346          dt = self%bin_end_time - self%previous_time
347          w = dt / (time - self%previous_time)   ! weight for current time (leaving 1-w for previous time)
348          _BEGIN_OUTER_VERTICAL_LOOP_
349             self%history(_PREARG_HORIZONTAL_LOCATION_ self%icurrent) = self%history(_PREARG_HORIZONTAL_LOCATION_ self%icurrent) &
350                + ((1._rke - 0.5_rke * w) * self%previous_value _INDEX_HORIZONTAL_LOCATION_ + 0.5_rke * w * value _INDEX_HORIZONTAL_LOCATION_) &
351                * dt / self%period
352          _END_OUTER_VERTICAL_LOOP_
353
354          if (self%complete) then
355             ! We already had a complete history (bins covering the full window size). Add the newly full bin, subtract the oldest bin
356             _BEGIN_OUTER_VERTICAL_LOOP_
357                self%last_exact_mean _INDEX_HORIZONTAL_LOCATION_ = self%last_exact_mean _INDEX_HORIZONTAL_LOCATION_ &
358                   - self%history(_PREARG_HORIZONTAL_LOCATION_ self%ioldest) + self%history(_PREARG_HORIZONTAL_LOCATION_ self%icurrent)
359             _END_OUTER_VERTICAL_LOOP_
360          elseif (self%icurrent == self%n) then
361             ! We just completed our history. Create the mean by summing all filled bins.
362             do ibin = 1, self%n
363                _BEGIN_OUTER_VERTICAL_LOOP_
364                   self%last_exact_mean _INDEX_HORIZONTAL_LOCATION_ = self%last_exact_mean _INDEX_HORIZONTAL_LOCATION_ &
365                      + self%history(_PREARG_HORIZONTAL_LOCATION_ ibin)
366                _END_OUTER_VERTICAL_LOOP_
367             end do
368             self%complete = .true.
369          end if
370
371          ! Update previous time and value to match end of current bin
372          self%previous_time = self%bin_end_time
373          _BEGIN_OUTER_VERTICAL_LOOP_
374             self%previous_value _INDEX_HORIZONTAL_LOCATION_ = (1._rke - w) * self%previous_value _INDEX_HORIZONTAL_LOCATION_ + w * value _INDEX_HORIZONTAL_LOCATION_
375          _END_OUTER_VERTICAL_LOOP_
376
```

```
377          ! Move to next bin: update indices, end time and empty newly current bin
378          self%icurrent = self%ioldest
379          self%ioldest = self%ioldest + 1
380          if (self%ioldest > self%n + 1) self%ioldest = 1
381          self%bin_end_time = self%bin_end_time + dt_bin
382          _BEGIN_OUTER_VERTICAL_LOOP_
383             self%history(_PREARG_HORIZONTAL_LOCATION_ self%icurrent) = 0
384          _END_OUTER_VERTICAL_LOOP_
385       end do
386
387       ! Compute average of previous and current value, multiply by time difference, pre-divide by window size, and add
     to current bin.
388       _BEGIN_OUTER_VERTICAL_LOOP_
389          self%history(_PREARG_HORIZONTAL_LOCATION_ self%icurrent) = self%history(_PREARG_HORIZONTAL_LOCATION_ self%icu
    rrent) &
390              + 0.5_rke * (self%previous_value _INDEX_HORIZONTAL_LOCATION_ + value _INDEX_HORIZONTAL_LOCATION_) &
391              * (time - self%previous_time) / self%period
392       _END_OUTER_VERTICAL_LOOP_
393
394       if (self%complete) then
395          ! We have a full history covering at least one window size. Update the running mean.
396          ! The result is an approximation that assumes linear change over the period covered by the oldest bin.
397          _BEGIN_OUTER_VERTICAL_LOOP_
398             self%mean _INDEX_HORIZONTAL_LOCATION_ = self%last_exact_mean _INDEX_HORIZONTAL_LOCATION_ &
399                + self%history(_PREARG_HORIZONTAL_LOCATION_ self%icurrent) &
400                - self%history(_PREARG_HORIZONTAL_LOCATION_ self%ioldest) * (time - self%bin_end_time + dt_bin) / dt_bi
    n
401          _END_OUTER_VERTICAL_LOOP_
402       end if
403
404       ! Store current time and value to enable linear interpolation in subsequent call.
405       self%previous_time = time
406       _BEGIN_OUTER_VERTICAL_LOOP_
407          self%previous_value _INDEX_HORIZONTAL_LOCATION_ = value _INDEX_HORIZONTAL_LOCATION_
408       _END_OUTER_VERTICAL_LOOP_
409    end subroutine
410
411    subroutine horizontal_temporal_maximum_update(self, time, value _POSTARG_HORIZONTAL_LOCATION_RANGE_)
412       class (type_horizontal_temporal_maximum), intent(inout) :: self
413       real(rke),                                intent(in)    :: time
414       real(rke) _ATTRIBUTES_GLOBAL_HORIZONTAL_, intent(in)    :: value
415       _DECLARE_ARGUMENTS_HORIZONTAL_LOCATION_RANGE_
416
417       integer  :: ibin
418       real(rke) :: w
419       _DECLARE_HORIZONTAL_LOCATION_
420
421       ! Note that all array processing below uses explicit loops in order to respect
422       ! any limits on the active domain given by the _HORIZONTAL_LOCATION_RANGE_ argument.
423
424       if (self%icurrent == -1) then
425          ! Start of simulation
426          self%bin_end_time = time + self%period / self%n
427          self%icurrent = 1
428       end if
429
430       do while (time >= self%bin_end_time)
431          ! Update previous time and value to match end of current bin. For the latter, linearly interpolate to value a
    t end-of-bin time.
432          w = (self%bin_end_time - self%previous_time) / (time - self%previous_time)   ! weight for current time (leavi
    ng 1-w for previous time)
433          self%previous_time = self%bin_end_time
434          _BEGIN_OUTER_VERTICAL_LOOP_
435             self%previous_value _INDEX_HORIZONTAL_LOCATION_ = (1._rke - w) * self%previous_value _INDEX_HORIZONTAL_LOC
    ATION_ + w * value _INDEX_HORIZONTAL_LOCATION_
436          _END_OUTER_VERTICAL_LOOP_
437
438          ! Complete the current bin by taking the maximum over its previous value and the value at end-of-boin time.
439          _BEGIN_OUTER_VERTICAL_LOOP_
440             self%history(_PREARG_HORIZONTAL_LOCATION_ self%icurrent) = max(self%history(_PREARG_HORIZONTAL_LOCATION_ s
    elf%icurrent), &
441                  self%previous_value _INDEX_HORIZONTAL_LOCATION_)
442          _END_OUTER_VERTICAL_LOOP_
443
444          self%complete = self%complete .or. self%icurrent == self%n
445          if (self%complete) then
446             ! We have a complete history - compute the maximum over all bins
447             _BEGIN_OUTER_VERTICAL_LOOP_
448                self%maximum _INDEX_HORIZONTAL_LOCATION_ = self%history(_PREARG_HORIZONTAL_LOCATION_ 1)
449             _END_OUTER_VERTICAL_LOOP_
450             do ibin = 2, self%n
451                _BEGIN_OUTER_VERTICAL_LOOP_
452                   self%maximum _INDEX_HORIZONTAL_LOCATION_ = max(self%maximum _INDEX_HORIZONTAL_LOCATION_,  &
453                      self%history(_PREARG_HORIZONTAL_LOCATION_ ibin))
454                _END_OUTER_VERTICAL_LOOP_
455             end do
456          end if
457
458          ! Move to next bin: update indices, end time and set maximum of newly current bin to current value (at start
    of bin)
459          self%icurrent = self%icurrent + 1
460          if (self%icurrent > self%n) self%icurrent = 1
461          self%bin_end_time = self%bin_end_time + self%period / self%n
462          _BEGIN_OUTER_VERTICAL_LOOP_
463             self%history(_PREARG_HORIZONTAL_LOCATION_ self%icurrent) = self%previous_value _INDEX_HORIZONTAL_LOCATION_

464          _END_OUTER_VERTICAL_LOOP_
465       end do
```

```
466
467       ! Update the maximum of the current bin
468       _BEGIN_OUTER_VERTICAL_LOOP_
469          self%history(_PREARG_HORIZONTAL_LOCATION_ self%icurrent) = max(self%history(_PREARG_HORIZONTAL_LOCATION_ self
    %icurrent), &
470              value _INDEX_HORIZONTAL_LOCATION_)
471       _END_OUTER_VERTICAL_LOOP_
472
473       ! Store current time and value to enable linear interpolation in subsequent call.
474       self%previous_time = time
475       _BEGIN_OUTER_VERTICAL_LOOP_
476          self%previous_value _INDEX_HORIZONTAL_LOCATION_ = value _INDEX_HORIZONTAL_LOCATION_
477       _END_OUTER_VERTICAL_LOOP_
478    end subroutine
479
480 end module fabm_expressions
```