A short introduction to the MPAS framework

Matt Hoffman
5/16/2022

MPAS History

- Spearheaded by Todd Ringler (LANL) and Bill Skamrock (NCAR) following development of TRSK discretization scheme for geophysical fluids (2010)
- Vision: shared framework for building geophysical fluid models using unstructured, global, variable resolution Voronoi grids
- Initially: MPAS-Shallow Water, MPAS-Ocean (LANL), MPAS-Atmosphere (NCAR)
- Later: MPAS-Seaice, MPAS-Land Ice (now MALI)

ELSEVIER

Journal of Computational Physics
Volume 229, Issue 9, 1 May 2010, Pages 3065-3090

A unified approach to energy conservation and potential vorticity dynamics for arbitrarily-structured C-grids

T.D. Ringler [a], J. Thuburn [b], J.B. Klemp [c], W.C. Skamarock [c]

[a] Theoretical Division, Los Alamos National Laboratory, Los Alamos, NM 87545, USA
[b] Mathematics Research Institute, School of Engineering, Computing and Mathematics, University of Exeter, Exeter EX4 4QF, UK
[c] National Center for Atmospheric Research, Boulder, CO 80307, USA

MPAS Repository history

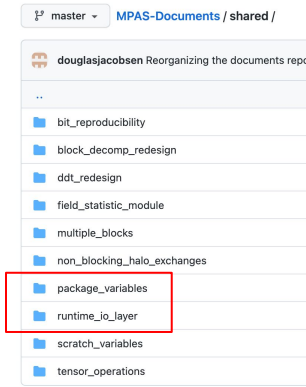
- Original svn repo: https://github.com/MPAS-Dev/MPAS-Legacy
  - First commit: Mar 4, 2010
- First git repo (originally private): https://github.com/MPAS-Dev/MPAS
  - I believe contains all the svn history
- Second git repo (entirely public): https://github.com/MPAS-Dev/MPAS-Model
  - Continuous git history with previous repo
  - Was a git submodule within E3SM (main motivation for going public)
- E3SM repo: https://github.com/E3SM-Project/E3SM
  - Transitioned May 17, 2021
  - Older history not accessible - refer to previous repo
  - Framework (including standalone Makefile) as a ‘component’ at: components/mpas-framework/
  - MPAS-Ocean, MPAS-Seaice, MPAS-Albany-Land Ice each have separate components
  - MALI development primarily in a fork: https://github.com/MALI-Dev/E3SM

MPAS Framework history

- Shared framework for data structures, mesh operations, parallelization, i/o, configuration parsing, timekeeping, physics-agnostic operations
- Primary authors:
  - Doug Jacobsen (LANL, now at Google)
  - Michael Duda (NCAR)
- Desire to make use of modern software features (templating, operator overloading, classes) but still use Fortran due to familiarity for domain experts
  - → many custom implementations, resulting in challenges for long term maintenance, ‘unusual’ Fortran code, mixed language code, etc.
- Desire to be very general - features any geophysical ‘core’ could use
  - → some features are overly general for any given physics application and perhaps more cumbersome than necessary for a specific application
- Currently in Framework:
  - 72,781 lines of Fortran code
  - 1845 lines of include files
  - 9554 lines of C code

Documentation

- Original documentation repo: https://github.com/MPAS-Dev/MPAS-Documents
- User's Guide: https://github.com/MPAS-Dev/MPAS-Documents/tree/master/users_guide
- Developer's Guide (Nov. 2013): https://github.com/MPAS-Dev/MPAS-Documents/blob/master/developers_guide/MPAS-DevelopersGuide.pdf
- Framework design documents: https://github.com/MPAS-Dev/MPAS-Documents/tree/master/shared



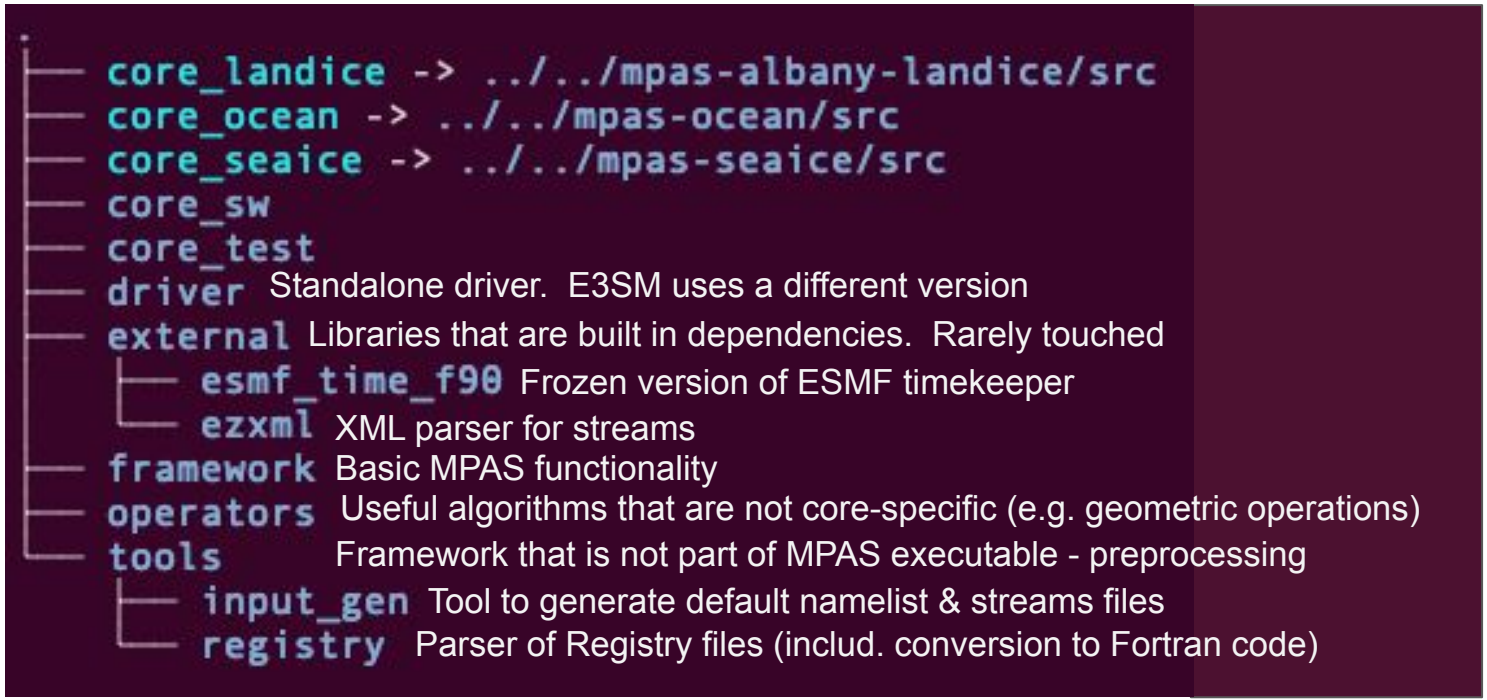
Development guidelines

As developers of MPAS, we attempt to make the code look as uniform as we can across the entire code-base. In order to enforce this, there are a set of guidelines developers should follow.

- Each core has a name, and an abbreviation. For example, the shallow water core is called sw and it's abbreviation is sw, but the ocean core is called ocean and it's abbreviation is ocn.
- All subroutines should be named in a manner which prevents namespace conflicts.
  Shared functions/subroutines are simply named mpas_subroutine_name.
  Core specific functions/subroutines are named mpas_abbrev_subroutine_n
  (where abbrev is replaced with the cores abbreviation).
  e.g. mpas_atm_time_integration
- Subroutine names should all be lower case, with underscores in place of spaces (e.g. see above).
- Variable names should be mixed case (e.g. cellsOnCell rather than cells_on_cell).
- In general, variable names should be self-descriptive (e.g. nCells rather than n).
- Subroutines and modules should be appropriately documented. Shared portions of MPAS code use doxygen comments, but core developers are free to decide what method of documenting they prefer.
- Development of shared parts of MPAS need reviews from multiple core maintainers prior to a merge.
- Development within a core should be approved by other core developers before being merged into that core.
- Development within a core should follow the practices of that core's developer group, for documentation etc.
- Core related testing is the responsibility of that core's maintainers/developers.

Framework structure

components/mpas-framework/src



Registry:

- XML file defining model dimensions, namelist options, stream definitions, variables
- parser converts to Fortran (and some C) code

Data structures:

- var_struct - grouping of model variables

```
<var_struct name="haloExchTest" time_levs="1">
        <var name="cellPersistReal5D" type="real" dimensions="TWO TWO vertexDegre
        <var name="cellPersistReal4D" type="real" dimensions="TWO vertexDegree ma
        <var name="cellPersistReal3D" type="real" dimensions="vertexDegree maxEdg
        <var name="cellPersistReal2D" type="real" dimensions="maxEdges nCells"/>
```

  - Var_array

```
        <var_struct name="mesh" time_levs="1">
                <var_array name="arrTest" type="real" dimensions="nCells" missing_value="FILLVAL">
                        <var name="test1" units="units1" description="Test var 1 in a var_array" array_group="arrTest
"/>
                        <var name="test2" units="units2" description="Test var 2 in a var_array" array_group="arrTest
"/>
                </var_array>
                <var name="latCell"                              type="real"     dimensions="nCells" description="Latitu
de of cell centers" units="radians" missing_value="-999"/>
```

    - Var

```
<var name="xCell"                                type="real"     dimensions="nCells"/>
```

Registry

```
<var_struct name="state" time_levs="2">
        <var_array name="tracers" type="real" dimensions="nVertLevels nCells Time">
                <var name="temperature" array_group="dynamics" streams="iro" units="degrees Celsius"
                    description="potential temperature"
                />
        />
        <var name="layerThickness" type="real" dimensions="nVertLevels nCells Time" streams="iro" units="m"
            description="Thickness of layer"
        />
/>
```

Allowable attributes defined in: src/tools/registry/Registry.xsd

- dimensions
- nml records & options
- streams
- var_structs , var_arrays, vars
  - **name**
  - **type**
  - **dimensions**
  - name_in_code
  - units
  - description
  - default_value
  - persistence
  - packages
  - (time_levs, array_group)

Data structures

Internal MPAS code:

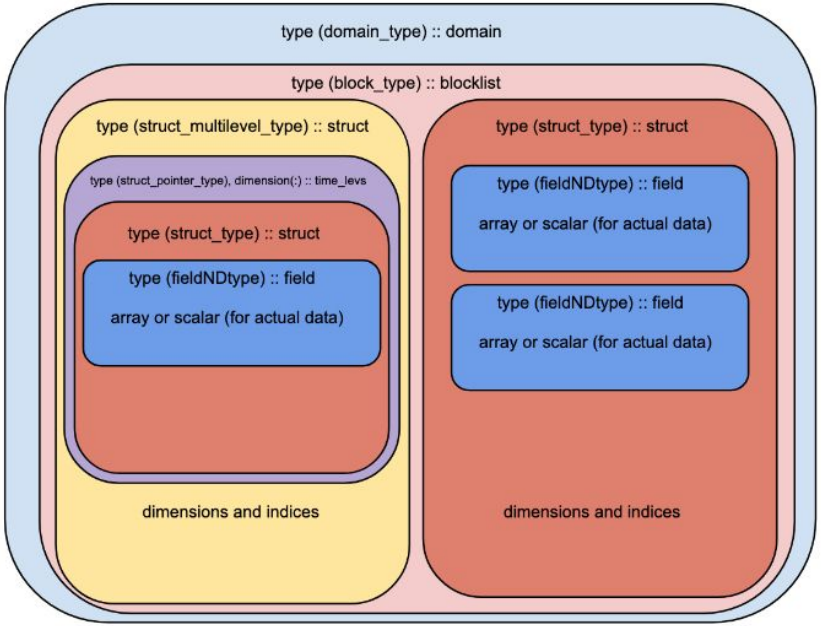


Figure 4.1: Visual diagram of MPAS derived data type layout.

As the blocklist structure is a linked list of blocks, one can iterate over the list of blocks in the following manner.

```
type (block_type), pointer :: block_ptr

block_ptr => domain % blocklist
do (while(associated(block_ptr))
        ... do stuff on block ...
        block_ptr => block_ptr % next
end do
```

Data structures: Internal MPAS code: ‘pools’

- similar to a class/derived type
  - each data structure has attributes and methods, child members, etc.
  - implemented from scratch with a lot of linked lists
  -

```
1519        !-----------------------------------------------------------------
1520        ! local variables
1521        !-----------------------------------------------------------------
1522        ! Pools pointers
1523        type (mpas_pool_type), pointer :: geometryPool
1524        type (mpas_pool_type), pointer :: hydroPool
1525        real (kind=RKIND), pointer :: rhoi, rhoo
1526        real (kind=RKIND), dimension(:), pointer :: thickness
1527        real (kind=RKIND), dimension(:), pointer :: waterPressure
1528        real (kind=RKIND), dimension(:), pointer :: bedTopography
1529        real (kind=RKIND), dimension(:), pointer :: hydropotentialBase
1530        real (kind=RKIND), dimension(:), pointer :: waterThickness
1531        real (kind=RKIND), dimension(:), pointer :: hydropotential
1532        real (kind=RKIND), dimension(:), pointer :: effectivePressure
1533        integer, dimension(:), pointer :: cellMask
1534        real (kind=RKIND), pointer :: config_sea_level
1535
1536        err = 0
1537
1538        ! Get pools things
1539        call mpas_pool_get_subpool(block % structs, 'hydro', hydroPool)
1540        call mpas_pool_get_subpool(block % structs, 'geometry', geometryPool)
1541
1542        call mpas_pool_get_config(liConfigs, 'config_ice_density', rhoi)
1543        call mpas_pool_get_config(liConfigs, 'config_sea_level', config_sea_level)
1544        call mpas_pool_get_config(liConfigs, 'config_ocean_density', rhoo)
1545
1546        call mpas_pool_get_array(hydroPool, 'effectivePressure', effectivePressure)
1547        call mpas_pool_get_array(geometryPool, 'thickness', thickness)
1548        call mpas_pool_get_array(hydroPool, 'waterPressure', waterPressure)
1549        call mpas_pool_get_array(geometryPool, 'bedTopography', bedTopography)
1550        call mpas_pool_get_array(hydroPool, 'hydropotentialBase', hydropotentialBase)
1551        call mpas_pool_get_array(hydroPool, 'waterThickness', waterThickness)
1552        call mpas_pool_get_array(hydroPool, 'hydropotential', hydropotential)
1553        call mpas_pool_get_array(geometryPool, 'cellMask', cellMask)
1554
1555        effectivePressure = rhoi * gravity * thickness - waterPressure
1556           ! < this should evalute to 0 for floating ice if Pw set correctly there.
1557        where (.not. li_mask_is_grounded_ice(cellmask))
1558           effectivePressure = 0.0_RKIND  ! zero effective pressure where no ice to av
1559        end where
1560
1561        hydropotentialBase = rho_water * gravity * bedTopography + waterPressure
1562        ! This is still correct under ice shelves/open ocean because waterPressure has
1563        ! Note this leads to a nonuniform hydropotential at sea level that is a functi
1564        ! That is what we want because we use this as a boundary condition on the subg
1565        ! and we want the subglacial system to feel the pressure of the ocean column a
1566
1567        ! hydropotential with water thickness
1568        hydropotential = hydropotentialBase + rho_water * gravity * waterThickness
```

define pointer variables
as destinations for pools
retrievals

mpas_pool_get_subpool to retrieve structs

mpas_pool_get_config to retrieve nl options

mpas_pool_get_array to retrieve actual arrays

Notes:

- pool routines are case sensitive even though Fortran is not!
- typos/faulty retrievals commands cannot be detected at compile time, only at run time! (major limitation of the pools data structure)
- Compiling with DEBUG=true necessary to get useful error message

mpas_pool_types.inc

```
integer, parameter :: MPAS_POOL_TABLE_SIZE = 128

integer, parameter :: MPAS_POOL_SILENT = 1001, &
                      MPAS_POOL_WARN   = 1002, &
                      MPAS_POOL_FATAL  = 1003

integer, parameter :: MPAS_POOL_FIELD     = 1004, &
                      MPAS_POOL_CONFIG    = 1005, &
                      MPAS_POOL_DIMENSION = 1006, &
                      MPAS_POOL_SUBPOOL   = 1007, &
                      MPAS_POOL_PACKAGE   = 1008

integer, parameter :: MPAS_POOL_REAL      = 1009, &
                      MPAS_POOL_INTEGER   = 1010, &
                      MPAS_POOL_LOGICAL   = 1011, &
                      MPAS_POOL_CHARACTER = 1012

type mpas_pool_data_type
   integer :: contentsType
   integer :: contentsDims
   integer :: contentsTimeLevs

   ! For storing fields
   type (field0DReal), pointer :: r0 => null()
   type (field1DReal), pointer :: r1 => null()
   type (field2DReal), pointer :: r2 => null()
   type (field3DReal), pointer :: r3 => null()
   type (field4DReal), pointer :: r4 => null()
   type (field5DReal), pointer :: r5 => null()
   type (field0DReal), dimension(:), pointer :: r0a => null()
   type (field1DReal), dimension(:), pointer :: r1a => null()
   type (field2DReal), dimension(:), pointer :: r2a => null()
   type (field3DReal), dimension(:), pointer :: r3a => null()
   type (field4DReal), dimension(:), pointer :: r4a => null()
   type (field5DReal), dimension(:), pointer :: r5a => null()
   type (field0DInteger), pointer :: i0 => null()
   type (field1DInteger), pointer :: i1 => null()
   type (field2DInteger), pointer :: i2 => null()
   type (field3DInteger), pointer :: i3 => null()
   type (field0DInteger), dimension(:), pointer :: i0a => null()
   type (field1DInteger), dimension(:), pointer :: i1a => null()
   type (field2DInteger), dimension(:), pointer :: i2a => null()
   type (field3DInteger), dimension(:), pointer :: i3a => null()
   type (field0DChar), pointer :: c0 => null()
   type (field1DChar), pointer :: c1 => null()
   type (field0DChar), dimension(:), pointer :: c0a => null()
   type (field1DChar), dimension(:), pointer :: c1a => null()
   type (field0DLogical), pointer :: l0 => null()
   type (field0DLogical), dimension(:), pointer :: l0a => null()
   type (mpas_pool_type), pointer :: p => null()

   ! For storing config options, dimensions, and packages
   integer, pointer :: simple_int => null()
   integer, dimension(:), pointer :: simple_int_arr => null()
   real(kind=RKIND), pointer :: simple_real => null()
   logical, pointer :: simple_logical => null()
   character(len=StrKIND), pointer :: simple_char => null()
end type mpas_pool_data_type
```

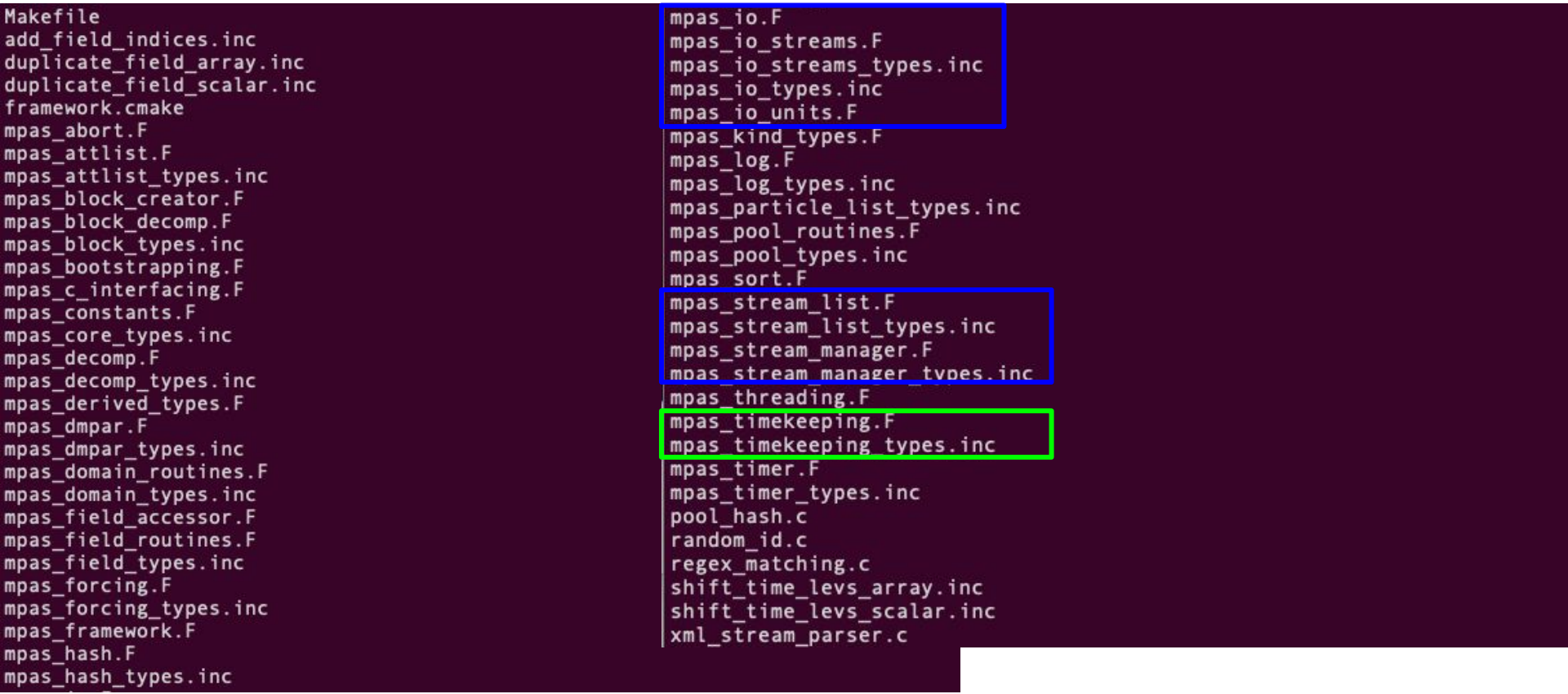
```
type mpas_pool_member_type
   character (len=StrKIND) :: key
   integer :: keyLen
   integer :: contentsType
   type (mpas_pool_data_type), pointer :: data => null()
   type (mpas_pool_member_type), pointer :: next => null()
   type (mpas_pool_member_type), pointer :: iteration_next => null()
   type (mpas_pool_member_type), pointer :: iteration_prev => null()
end type mpas_pool_member_type

type mpas_pool_head_type
   type (mpas_pool_member_type), pointer :: head => null()
end type mpas_pool_head_type

type mpas_pool_type
   integer :: size
   type (mpas_pool_head_type), dimension(:), pointer :: table => null()
   type (mpas_pool_member_type), pointer :: iterator => null()
   type (mpas_pool_member_type), pointer :: iteration_head => null()
   type (mpas_pool_member_type), pointer :: iteration_tail => null()
end type mpas_pool_type

type mpas_pool_iterator_type
   character (len=StrKIND) :: memberName
   integer :: memberType
   integer :: dataType
   integer :: nDims
   integer :: nTimeLevels
end type mpas_pool_iterator_type

type mpas_pool_field_info_type
   integer :: fieldType
   integer :: nDims
   integer :: nTimeLevels
   integer :: nHaloLayers
   logical :: isActive
end type mpas_pool_field_info_type
```

mpas_field_types.inc

```
! Derived type for storing fields
type field1DReal

   ! Back-pointer to the containing block
   type (block_type), pointer :: block => null()

   ! Raw array holding field data on this block
   real (kind=RKIND), dimension(:), pointer :: array => null()

   ! Information used by the I/O layer
   character (len=StrKIND) :: fieldName
   character (len=StrKIND), dimension(:), pointer :: constituentNames => null()
   character (len=StrKIND), dimension(1) :: dimNames
   integer, dimension(1) :: dimSizes
   real (kind=RKIND) :: defaultValue
   real (kind=RKIND) :: missingValue
   logical :: isDecomposed
   logical :: hasTimeDimension
   logical :: isActive
   logical :: isVarArray
   logical :: isPersistent
   type (att_lists_type), dimension(:), pointer :: attLists => null()

   ! Pointers to the prev and next blocks for this field on this task
   type (field1DReal), pointer :: prev => null()
   type (field1DReal), pointer :: next => null()

   ! Halo communication lists
   type (mpas_multihalo_exchange_list), pointer :: sendList => null()
   type (mpas_multihalo_exchange_list), pointer :: recvList => null()
   type (mpas_multihalo_exchange_list), pointer :: copyList => null()
end type field1DReal
```

src/framework

- .F -> code
- .inc -> derived type definitions



MPAS Timekeeping

- xtime is most fundamental time variable
  - string of format: ‘YYYY-MM-DD_hh:m:ss’
- cores have defined something like ‘daysSinceStart’ as a real variable but not actually used by framework
- MPAS_Time_type (ESMF_Time) used for most actual time operations
- MPAS wraps and old, frozen version of ESMF timekeeper
  - (E3SM might use a more current version?)
  - handles Y,M,D,h,m,s, including different calendars (e.g. leap years) and overloaded mathematic operators
  - concept of ‘time’, ‘interval’
  - concept of ‘clock’, ‘alarm’
  - Addition of a CF compliant time variable may require adjusting timekeeping routines (or possibly updating the version of the ESMF timekeeper?)

mpas_timekeeping_types.inc

```
integer, parameter :: MPAS_MAX_ALARMS = 40

integer, parameter :: MPAS_NOW = 0, &
                      MPAS_START_TIME = 1, &
                      MPAS_STOP_TIME = 2

integer, parameter :: MPAS_FORWARD = 1, &
                      MPAS_BACKWARD = -1

integer, parameter :: MPAS_GREGORIAN = 0, &
                      MPAS_GREGORIAN_NOLEAP = 1, &
                      MPAS_360DAY = 2

type MPAS_Time_type
   type (ESMF_Time) :: t
end type

type MPAS_TimeInterval_type
   type (ESMF_TimeInterval) :: ti
end type

type MPAS_Alarm_type
   character (len=ShortStrKIND) :: alarmID
   logical :: isRecurring
   logical :: isSet
   type (MPAS_Time_type) :: ringTime
   type (MPAS_Time_type) :: prevRingTime
   type (MPAS_TimeInterval_type) :: ringTimeInterval
   type (MPAS_Alarm_type), pointer :: next => null()
end type

type MPAS_Clock_type
   integer :: direction
   integer :: nAlarms
   type (ESMF_Clock) :: c
   type (MPAS_Alarm_type), pointer :: alarmListHead => null()
```

mpas_timekeeping.F

```
module mpas_timekeeping

   use mpas_kind_types
   use mpas_derived_types
   use mpas_dmpar
   use mpas_threading
   use mpas_log

   use ESMF
```

```
subroutine mpas_timekeeping_init
subroutine mpas_timekeeping_finalize
subroutine mpas_timekeeping_set_year_width!}}}
subroutine mpas_create_clock
subroutine mpas_destroy_clock
subroutine mpas_set_clock_direction
subroutine mpas_set_clock_timestep
subroutine mpas_advance_clock
subroutine mpas_set_clock_time
subroutine mpas_add_clock_alarm
subroutine mpas_remove_clock_alarm
subroutine mpas_minimum_alarm_interval
subroutine mpas_print_alarm
subroutine mpas_get_clock_ringing_alarms
subroutine mpas_reset_clock_alarm
subroutine mpas_adjust_alarm_to_reference_time
subroutine mpas_calibrate_alarms
subroutine mpas_set_time
subroutine mpas_get_time
subroutine mpas_set_timeInterval
subroutine mpas_get_timeInterval
subroutine mpas_interval_division
subroutine mpas_split_string
subroutine mpas_get_month_day
subroutine mpas_expand_string!}}}
```