Other Fortran Data Interfaces

These derived types contain data that is not in model or observation space.

Locations

```
type :: locations
contains
  procedure(loc_create) :: create  ! Constructor
  procedure(loc_delete) :: delete  ! Destructor
  procedure(loc_print)  :: print   ! Prints human readable info
end type locations

interface
  subroutine loc_create(self, obspace, bgn, end)
    type(locations), intent(inout) :: self
    type(obs_space), intent(in) :: obspace
    type(datetime), intent(in)  :: bgn  ! Restrict to obs within a time slot
    type(datetime), intent(in)  :: end  ! Restrict to obs within a time slot
  end subroutine loc_create
end interface
```

As a first implementation, a Locations type could contain two obs_data entities (or something based on the same structure) that can be filled with latitudes and longitudes read from an obs_space object. In the longer term, more complex locations will be needed (slanted profile, satellite footprint, GPSRO line of sight...). A random method might be useful for testing of interpolations, however, it has to be compatible with the type of application being tested (global, LAM, ocean, etc...). A quick and dirty implementation could be to wrap the whole obs_space object.

Variables

```
type :: variables
contains
  procedure(var_create) :: create  ! Constructor
  procedure(var_delete) :: delete  ! Destructor
  procedure(var_print)  :: print   ! Prints human readable info
end type variables

interface
  subroutine var_create(self, config)
    type(variables), intent(inout) :: self
    type(config), intent(in)       :: config
  end subroutine var_create
end interface
```

Variables can be implemented in many ways. The cleanest would be an actual list of varaibles (for example based on the NetCDF CF or GRIB conventions). It could also be a much more model specific derived type containing flags or anything necessary to know what fields are required by the model if the state is going to be used as an initial condition, or what variables are needed by an observation operator when the Variables is used as an input to the interpolations.

Fields at Locations

```
type :: fields_at_locations
contains
  procedure :: create       ! Constructor
  procedure :: delete       ! Destructor
  procedure :: zero         ! Fill woth zeros
  procedure :: random       ! Fill with random values
  procedure :: dot_product  ! Dot product
  procedure :: print        ! Prints human readable info
end type fields_at_locations

interface
  subroutine create(self, obspace, bgn, end, geom, vars)
    type(fields_at_locations), intent(inout) :: self
    type(obs_space), intent(in) :: obspace ! Obs metadata
    type(datetime), intent(in)  :: bgn     ! Restrict to obs within a time slot?
    type(datetime), intent(in)  :: end     ! Restrict to obs within a time slot?
    type(geometry), intent(in)  :: geom    ! Only vertical geometry should be needed
    type(variables), intent(in) :: vars    ! Variables needed by obs opertor
  end subroutine create
end interface
```

At first, `fields_at_locations` could be seen as a state with an unstructured geometry. However, it cannot be substituted for a state as initial condition for a forecast. In many applications, it will contain values distributed geographically or an unstructured grid, but also distributed in time so it does not have a valid time like a state would have. For these reasons, it cannot be a sub-class of state.

Configuration

Most constructors take a configuration parameter (type config). This represents user input, typically what would be read from namelists in traditional fortran. Instances could contain actual values to be used inside the routines, or at least information for lower level fortran code to read in the proper namelists. When writing a constructor (or other method that need user configuration), one cannot assume that only one instance will be created or that all instances will be created from the same parameter. That information can only come from higher level code, even if that information is only the name of a file where to read namelists from.