Observation Space Fortran Interfaces

The classes above manipulate data in observation space. Three classes will be used to handle this data. The first class represent the "observation space". It can be interpreted as a mirror image in the observation world of what the grid definition is in model/state space. As such, it will be passed as input to the constructors of objects in observation space. Its role is however more complex than the grid definition in model space because metadata is associated to observations. The obs_space objects will handle the metadata for objects in the observation space. In concrete terms, this object will encapsulate the main observation data structure. However, it will not contain all objects in this observation space. For example, all intermediate quantities computed in the data assimilation process, such as o-b, o-a, etc... will exist as independent objects (of type obs_data or obs_vector).

From an abstract point of view, the obs_space class has a very simple interface, although more functionality might be needed at lower level.

Observation Space

```
type :: obs_space
contains
  procedure(obs_create) :: create  ! Constructor
  procedure(obs_delete) :: delete  ! Destructor
  procedure(obs_print)  :: print   ! Prints human readable info
end type obs_space
```

In its first implementation, the obs_space structure can be the linked-list that already exists in the GSI, but where "columns" created in the DA system are removed and stored in obs_data objects.

Data in observation space is stored in objects of class obs_data. Interpreting the overall observation data structure as a generalized table, an instance of type obs_data would hold one column of that table (but not a column of metadata). Such objects can be manipulated as required by high level algorithms: one instance for y^{obs}, one for H(x), etc... Continuing with the analogy with the model space, an obs_data instance is the equivalent of a state instance. At the interface level, it will have similar methods for read and write, although there implementation will most likely be very different: it is most likely that the read will access the obs_space object to read values while the write will add a column to the obs_space object. Actual I/O will most likely happen in the constructor (read) and destructor (write) of the obs_space object but this is not visible from the users point of view. For all this to be possible, the obs_data and/or obs_space implementations will need to keep enough information to match the values in the obs_data column with the appropriate "rows" in the obs_space object.

Observation Data

```
type :: obs_data
contains
  procedure(obsdata_create) :: create  ! Constructor
  procedure(obsdata_delete) :: delete  ! Destructor
  procedure(obsdata_delete) :: read    ! Read data
  procedure(obsdata_delete) :: write   ! Write data
  procedure(obsdata_print)  :: print   ! Prints human readable info
end type obs_data
```

Finally, the obs_vector class is an extension of the obs_data class with added functionality for linear algebra.

Observation Vector

```
type, extends(obs_data) :: obs_vector
contains
  procedure(obsvec_add)  :: add         ! Add vectors
  procedure(obsvec_sub)  :: sub         ! Subtract vectors
  procedure(obsvec_mult) :: mult        ! Multiply vector by scalar
  procedure(obsvec_zero) :: zero        ! Set vector to zero
  procedure(obsvec_dotp) :: dot_product ! Dot product
end type obs_vector
```

Other classes

Additional classes for handling observation bias correction (or other observation space auxiliary variables) will be added.