GetValues Class Fortran

Code is in feature/getvalues of fv3-jedi

GetValues Base Type

```
implicit none
private
public :: fv3jedi_getvalues, fv3jedi_getvalues_base

type, abstract :: fv3jedi_getvalues_base
  integer              :: isc, iec, jsc, jec, npz, ngrid
  character(len=2048)  :: interp_method
  type(bump_type)      :: bump
  integer              :: nnear = 4
  type(unstrc_interp)  :: unsinterp
  type(fckit_mpi_comm) :: comm
  contains
    procedure, public :: create
    procedure, public :: delete
    procedure, public :: fill_geovals
    generic, public :: set_trajectory => fill_geovals
    generic, public :: fill_geovals_tl => fill_geovals
end type fv3jedi_getvalues_base
```

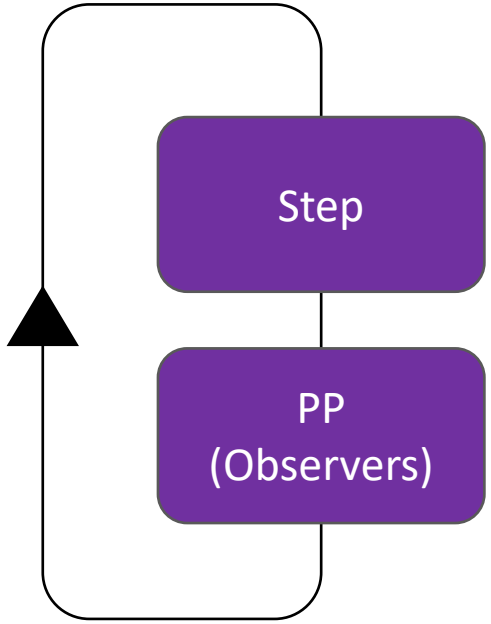
GetValues

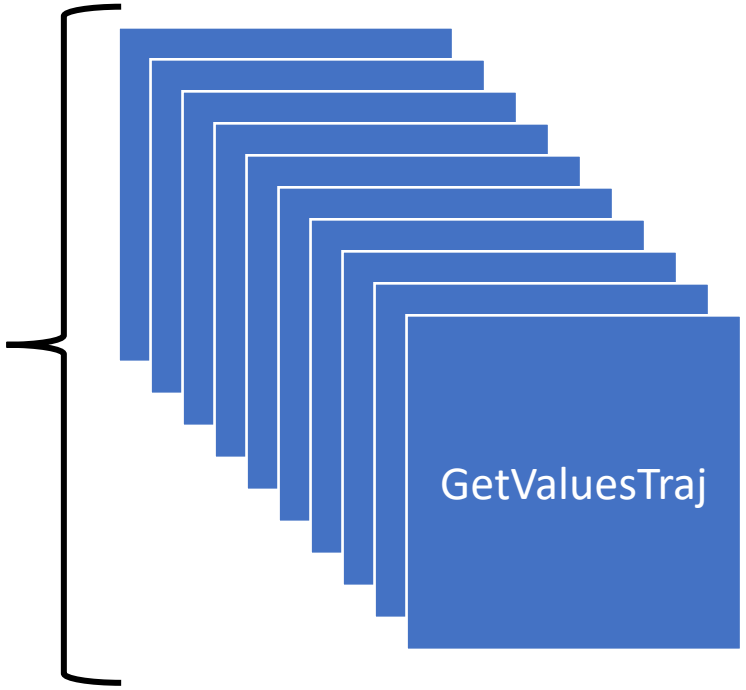
```
type, extends(fv3jedi_getvalues_base) :: fv3jedi_getvalues
end type fv3jedi_getvalues
```

LinearGetValues

```
type, extends(fv3jedi_getvalues_base) :: fv3jedi_lineargetvalues
  contains
    procedure, public :: fill_geovals_ad
end type fv3jedi_lineargetvalues
```

Old GetValues





Each GetValuesTraj object represented a snapshot of the window and only saw the locations in that sub window.

Interpolation weights could only be computed for the locations in the sub window, resulting in many interpolation weight creation calls.

For the new GetValues there is just one object that stores the trajectory for the entire window.

New GetValues

Create weights for all locations
(called once)

```
subroutine create(self, geom, locs)

class(fv3jedi_getvalues_base), intent(inout) :: self
type(fv3jedi_geom),            intent(in)    :: geom
type(ufo_locs),                intent(in)    :: locs

end subroutine create
```

Delete weights
(called once)

```
subroutine delete(self)

class(fv3jedi_getvalues_base), intent(inout) :: self

end subroutine delete
```

Apply weights and mask to
locations between t1 and t2
(called each timestep)

```
subroutine fill_geovals(self, geom, fields, t1, t2, locs, geovals)

class(fv3jedi_getvalues_base), intent(in) :: self
type(fv3jedi_geom),            intent(in)    :: geom
type(fv3jedi_field),           intent(in)    :: fields(:)
type(datetime),                intent(in)    :: t1
type(datetime),                intent(in)    :: t2
type(ufo_locs),                intent(in)    :: locs
type(ufo_geovals),             intent(inout) :: geovals

end subroutine fill_geovals
```

Convenience Pointers

Create weights

Optionally create BUMP interpolation weights:

```
self%interp_method = trim(geom%interp_method)
if (trim(self%interp_method) == 'bump') then
  call bump_init(geom, locs%nlocs, locs%lat, locs%lon, self%bump, 55555)
endif
```

Always create unstructured interpolation weights:

```
call self%unsinterp%create( geom%f_comm, self%nnear, 'barycent', &
                            self%ngrid, rad2deg*geom%lat_us, rad2deg*geom%lon_us, &
                            locs%nlocs, locs%lat, locs%lon )
```

Example with unstructured interpolation

```
do jlev = 1, field%npz
  n = 0
  do jj = field%jsc, field%jec
    do ji = field%isc, field%iec
      n = n + 1
      field_us(n) = field%array(ji, jj, jlev)
    enddo
  enddo

  if (.not. field%integerfield .and. trim(field%space)=='magnitude') then
    call self%unsinterp%apply(field_us, geovals_tmp)
  elseif (field%integerfield) then
    call unsinterp_integer_apply(self%unsinterp, field_us, geovals_tmp)
  elseif (trim(field%space)=='direction') then
    call unsinterp_nearest_apply(self%unsinterp, field_us, geovals_tmp)
  else
    call abor1_ftn("fv3jedi_getvalues_mod.fill_geovals: interpolation for this kind of "// &
                   "field is not supported. FieldName: "// trim(field%fv3jedi_name))
  endif
  geovals_all(1:locs%nlocs, jlev) = geovals_tmp(1:locs%nlocs)
enddo
```

Field to unstructured

Interpolate regular fields

Interpolate integer fields

Interpolate directional fields

Fill local GeoVaLs structure

Apply Mask:

ufo::Locations now provides a mask of locations within two times:

```
logical, allocatable :: time_mask(:)

! Get mask for locations in this time window
! -------------------------------------------
call ufo_locs_time_mask(locs, t1, t2, time_mask)
```

Fill section of GeoVaLs valid between two times

```
! Fill GeoVaLs relevant to this window
! ------------------------------------
do n = 1,locs%nlocs
  if (time_mask(n)) geovals%geovals(gv)%vals(1:field%npz, n) = geovals_all(n, 1:field%npz)
enddo
```

Variable Change

We need to move away from GetValues being responsible for variable changes. As well as breaking the separation of concerns it's not easily compatible with the new structure of GetValues, where there is one trajectory structure for all variables.

```
void GetValues::compute(const State & state, const util::DateTime & t1,
                        const util::DateTime & t2, ufo::GeoVaLs & geovals) const {
  oops::Log::trace() << "GetValues::fillGeovals starting" << std::endl;
  const util::DateTime * t1p = &t1;
  const util::DateTime * t2p = &t2;

  // Create state with geovals variables
  State stategeovalvars(*geom_, geovals.getVars(), state.validTime());
  model2geovals_->changeVar(state, stategeovalvars);

  // Fill GeoVaLs
  fv3jedi_getvalues_fill_geovals_f90(keyGetValues_, geom_->toFortran(),
                                     stategeovalvars.toFortran(), &t1p, &t2p, locs_.toFortran(),
                                     geovals.toFortran());

  oops::Log::trace() << "GetValues::fillGeovals done" << std::endl;
}
```

First create state with variables of the GeoVaLs and transform.

Once in the Fortran the code is generic and makes no reference to specific fields

```
private:
 std::unique_ptr<VarChaModel2GeoVaLs> model2geovals_;
```

LinearVariableChange

The linear variable change is a little more complex due to the time dependent trajectory:

```
private:
 const LinVarChaModel2GeoVaLs * getLinVarCha(const util::DateTime &) const;

 typedef std::map< util::DateTime, LinVarChaModel2GeoVaLs * >::iterator lvcIter;
 typedef std::map< util::DateTime, LinVarChaModel2GeoVaLs * >::const_iterator lvcIterCnst;

 std::map< util::DateTime, LinVarChaModel2GeoVaLs * > linearmodel2geovals_;
```

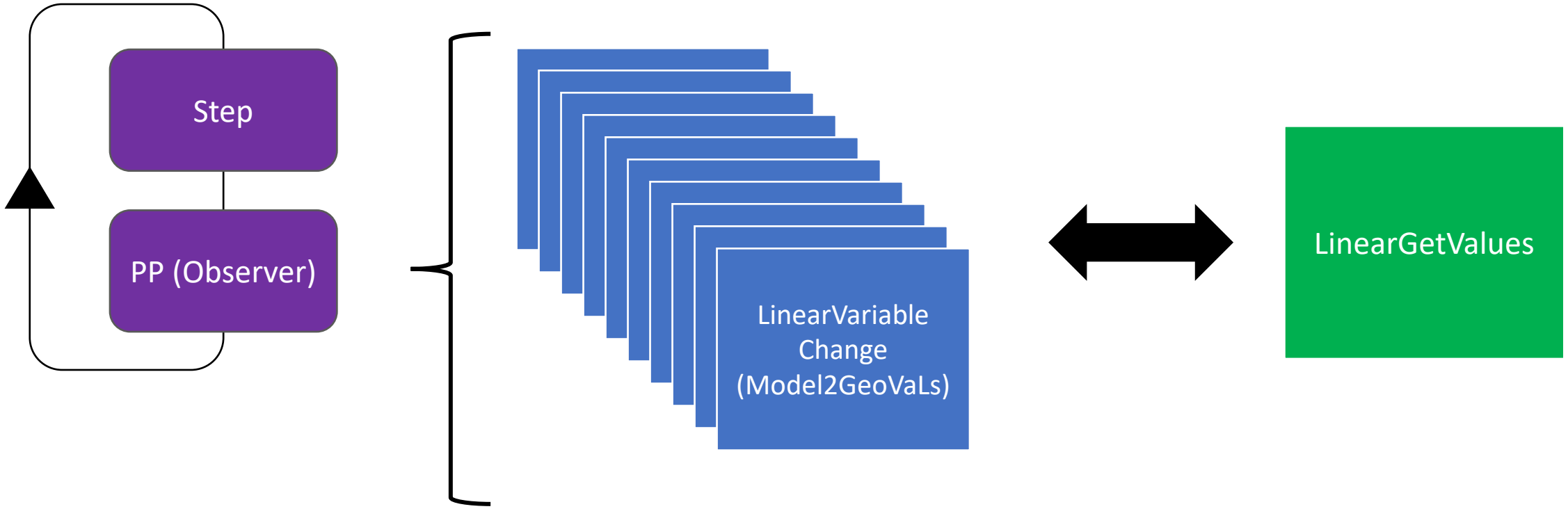
LinearGetValues.h defines a map between time and linear variable change object and provides iterators.

```
void LinearGetValues::setTrajectory(const State & state, const util::DateTime & t1,
                                    const util::DateTime & t2, ufo::GeoVaLs & geovals) {
  ...
  // Create the linear variable change object
  ASSERT(linearmodel2geovals_.find(t1) == linearmodel2geovals_.end());
  char sep = '.';
  eckit::LocalConfiguration dummyconfig(sep);
  LinVarChaModel2GeoVaLs * linearmodel2geovals = new LinVarChaModel2GeoVaLs(state, state, *geom_,
                                                                            dummyconfig);
  linearmodel2geovals_[t1] = linearmodel2geovals;

  fv3jedi_lineargetvalues_set_trajectory_f90(keyLinearGetValues_, geom_->toFortran(),
                                             stategeovalvars.toFortran(), &t1p, &t2p,
                                             locs_.toFortran(), geovals.toFortran());
}
```

Create the variable change in setTrajectory and add it to the map. Methods computeTL and computeAD can access the correct variable change using the iterator.

New GetValues



Increase in number of fields

- Now that we use variable changes we need to define the large number of potential fields that UFO has. In FV3-JEDI this is handled using the FieldsMetaData class

Data/fieldsets/ufo.yaml

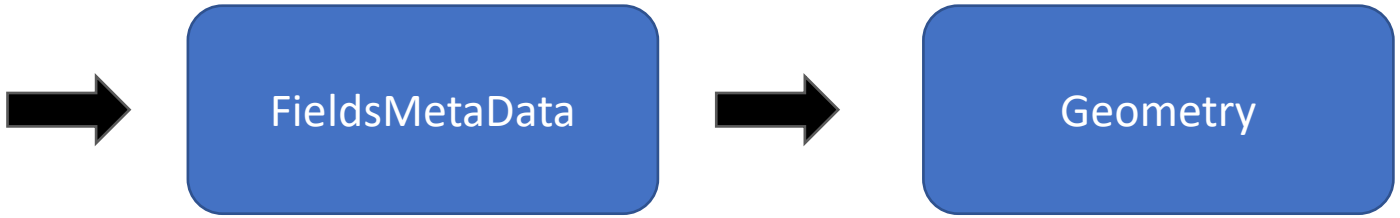
```yaml
Fields:
  - FieldName: geopotential_height
    Units: m
    Space: magnitude
    Levels: full

  - FieldName: surface_wind_from_direction
    Units: none
    Space: direction
    Levels: 1

  - FieldName: land_type_index
    Units: none
    Levels: 1
    Kind: integer

  - FieldName: humidity_mixing_ratio
    Units: 1
    Levels: full
    Tracer: true

    ....
```



Field constructor

```
do var = 1, vars%nvars()
fc=fc+1;
  fmd = geom%fields%get_field(trim(vars%variable(var)))
  call self%fields(fc)%allocate_field(geom%isc, geom%iec, geom%jsc, geom%jec, &
                                      fmd%levels, &
                                      short_name   = trim(fmd%field_io_name), &
                                      long_name    = trim(fmd%long_name), &
                                      fv3jedi_name = trim(fmd%field_name), &
                                      units        = fmd%units, &
                                      space        = trim(fmd%space), &
                                      staggerloc   = trim(fmd%stagger_loc), &
                                      tracer       = fmd%tracer, &
                                      integerfield = trim(fmd%array_kind)=='integer')
enddo
```

Summary

- Cleaner and likely faster weight generation for the interpolation.
- No more variable changes in the compute part of GetValues.
- GetValues is completely generic.
- Removed code duplication.
- Get values code from 2500 lines to 320 (get values) + 1200 (variable changes)

Future:

Some efficiency could be gained by now moving the definition of the variable change outside of the loop over observers. E.g. all the CRTM operators need the same variables so why repeat that calculation.

With the code constructed as demonstrated that change would be entirely in oops with just code removal from the model.

An interpolation apply method that can receive the mask could be beneficial as it could reduce communication/kdtree size.