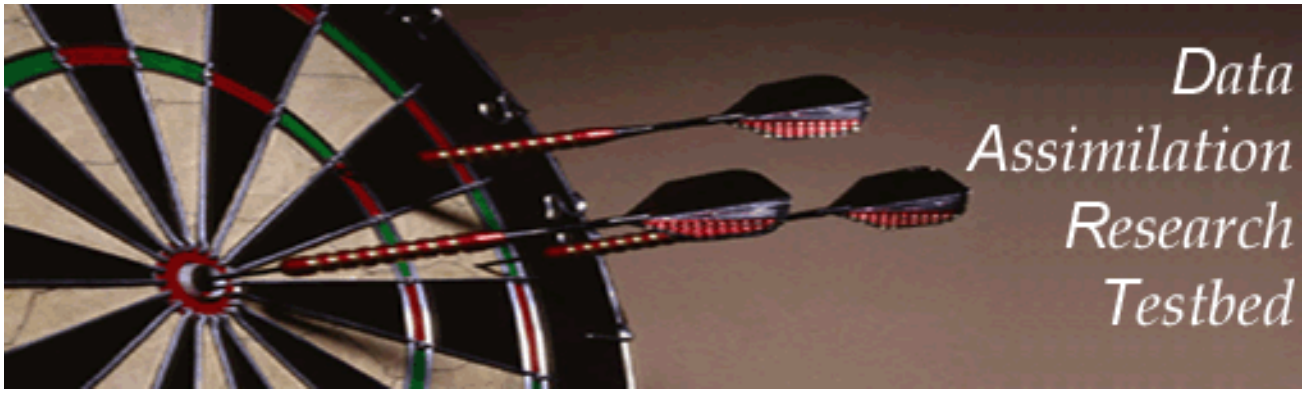
Data Assimilation Research Testbed Tutorial

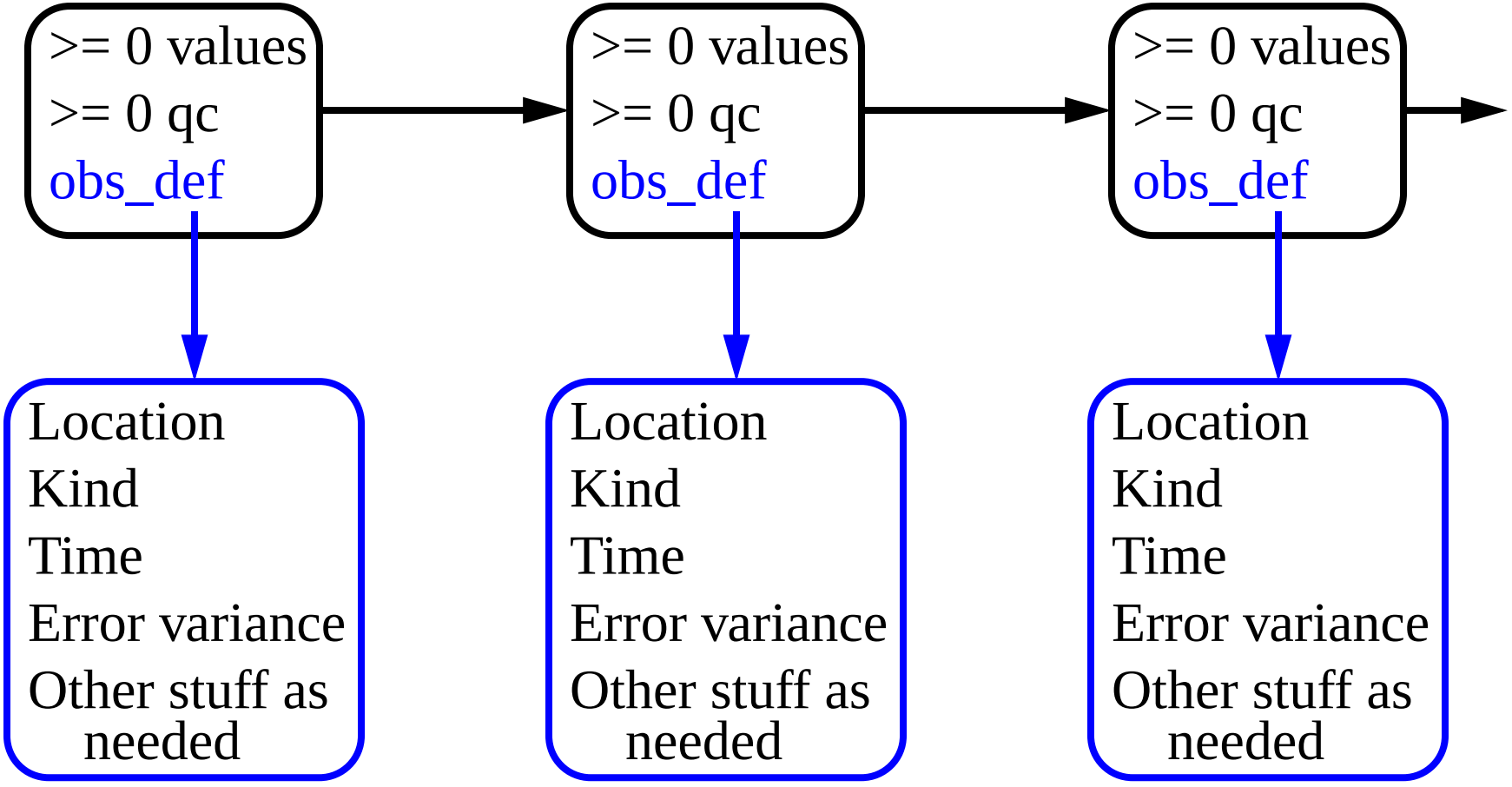


Section 17: Creating Observation Sequences

Version 1.0: June, 2005

Structure of an obs_sequence file:

Sequence with non-decreasing times in definitions.



Building Real observation sequences:

1. Interactive direct construction: program *create_obs_sequence*.
   Queries for information for each observation in turn.
   Enter kind, location, time, error variance, value, qc value(s).

   Often convenient to create an input file.
   Then redirect this file to standard input for *create_obs_sequence*.

2. Creating your own program.
   The obs_sequence module provides full set of interfaces to create.
   Example: translation from NCEP BUFR file format.
      Reads BUFR files, writes obs_sequence.

Creating Synthetic Observation Sequences (OSSEs):

Step 1: Create an observation sequence with no values.

A. Direct use of *create_obs_sequence*: no need to specify value for obs.
   OR...
B. Synthetic observing network fixed in time:
   1. First, use *create_obs_sequence* to specify observations in fixed network, all with time 0 days, 0 seconds.
   2. Use *create_fixed_network_seq* to specify times at which fixed network is observed.
   3. Times can be regularly or irregularly spaced.

Creating Synthetic Observation Sequences (OSSEs):

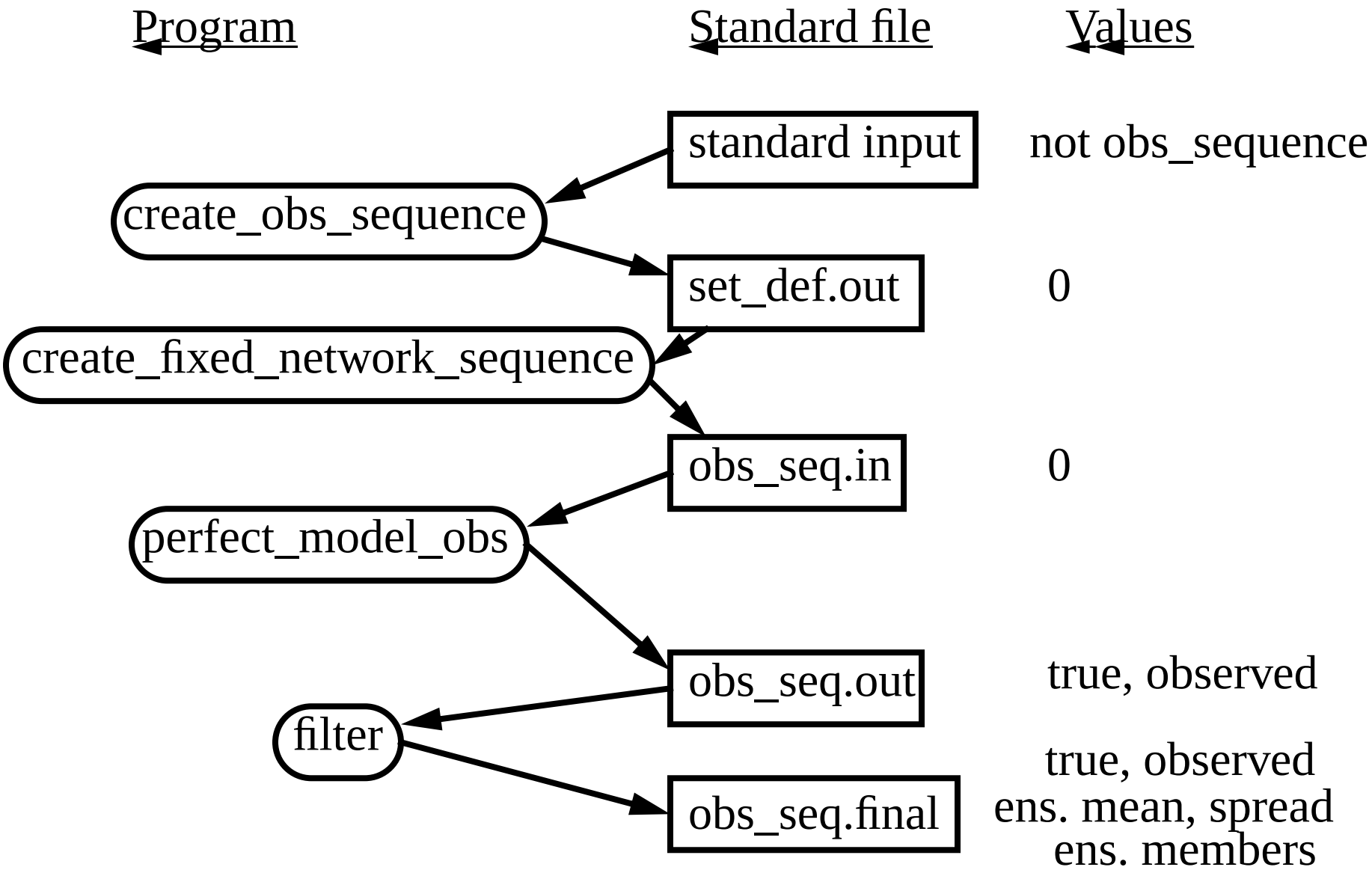
Step 2: Use perfect model_obs to add observed values:

1. Integrates model.
2. Applies forward operators to get ‘true’ observed values.
3. Adds sample from observational error to get observed value.
   (Output obs_sequence has 2 values for each observation).

Step 3: Run the filter:

Ensemble mean, spread, and individual ensemble members are added as values if requested (*filter_nml*, see section 16).

Creating Synthetic Observation Sequences (OSSEs):



Selecting set of observation definitions:

Limitations of F90 make this messy

Need to specify via name in namelist:

Kind of all observations to be assimilated;

Kind of all observations to be evaluated but not assimilated.

(Forward operators are computed and stored in obs_sequence).

List of available observation types found in *obs_def_mod.f90*.

(see declaration for obs_kind_info).

Specify in *obs_def_nml* using names:

```
&obs_def_nml
   assimilate_these_obs_types = 'raw_state_variable'
   evaluate_these_obs_types   = 'raw_state_1d_integral'/
```

Selecting set of observation definitions:

Another limitation of F90.

Have to preprocess obs_def_mod.F90 using program *preprocess*.

obs_def_mod.F90 marks blocks of conditionally compiled codes with:

```
#ifdef observation_kind_name
    <conditionally compiled code>
#endif
```

Program *preprocess* reads *obs_def_nml*, generates *obs_def_mod.f90*.

Only code required for requested observation kinds is compiled.

Procedure for setting up experiment with modified observation kinds.

1. Edit the *input.nml* to select kinds for assimilation and evaluation.

2. Run *preprocess*.

3. If any new observation kinds with complex forward operators are requested, add obs_def_??? modules for these kinds to path_names.

4. Remove any no longer used observation modules from path_names.

5. Do *mkmf* and *make* for all programs to be used.

This is least elegant part of DART: let us know if you have ideas.