pdb_file_name = None
    Input PDB file name (model: macromolecule + water ONLY)
external_da_pdb_file_name = None
    PDB file name with DA (ONLY). In this case the DA will not be built. Distance-based filtering will be done using model and sphere parameters (if defined).
reflection_file_name = None
    Fobs or Iobs data file name (with or w/o free-R flags)
data_labels = None
    Labels of reflections data (if multiple and ambiguous in input file)
refine = *occupancies *adp *sites
    Define what to refine for DA only (none of "macromolecule + water" parameters will be refined at all)
stop_reset_occupancies_at_macro_cycle = 5
    During the first stop_reset_occupancies_at_macro_cycle refinement macro-cycles: reset occupancies of DA to obey limits defined in "filter" scope. Zero value will disable the feature.
stop_reset_adp_at_macro_cycle = 5
    During the first stop_reset_adp_at_macro_cycle refinement macro-cycles: reset b-factors of DA to obey limits defined in "filter" scope. Zero value will disable the feature.
start_filtering_at_macro_cycle = 10
    Start applying DA filtering criteria (defined in "filter" scope) starting at macro-cycle number start_filtering_at_macro_cycle.
sphere {
  center = None
  radius = None
}
    Define sphere center and radius. The DA will be build inside of this sphere. If external DA used, then the sphere parameters (if defined) will be used for filtering only).
mode = build *build_and_refine
    DA mode: build only and exit, or build and refine. If external DA used then they will be refined and filtered out only (no new DA will be added)
high_resolution = None
    Filer input reflections by resolution.
low_resolution = None
    Filer input reflections by resolution.
output_file_name_prefix = None
    Prefix for all output files.
initial_occupancy = 0.1
    Start occupancy for newly added DA. Note, PDB file precision allows only two digits for occupancies; that is if occupancy is less than 0.005 it will appear as zero in output PDB file.
initial_b_factor = None
    Start B-factor for newly added DA. If not defined then the overall average is used.
atom_gap = 0.7
    Distance between grid nodes where DA are placed.
overlap_interval = 0.25
    DISABLED. NOT USE ANYWHERE. Hidden functionality is that multiple sets of DA can be added within one sphere, where each set is shifted w.r.t. another by overlap_interval distance. This creates a smaller effective atom_gap. To enable: the code will need some refactoring.
atom_type = H
    Chemical type of newly placed DA.
atom_name = " DA "
residue_name = DUM
scattering_table = *n_gaussian wk1995 it1992 neutron
number_of_refinement_cycles = 20

```
number_of_minimization_iterations = 25
filter {
  b_iso_min = 1.0
  b_iso_max = 100.0
  occupancy_min = 0.1
  occupancy_max = 10.0
  inside_sphere_scale = 1.2
        DA with distance larger than sphere.radius*inside_sphere_scale from sphere.center will be removed.
  da_model_min_dist = 1.
}
        DA filtering criteria.
```

See code for details: ../mmtbx/grow_density.py

All defaults are set to whatever worked best in some my limited tests.

---

phenix.grow_density workflow (very sketchy):

- Read in inputs (PDBs, reflections, parameters);
- Create new DA or just use user provided ones;
- Set up fmodel object – the Python object that holds on PDB structure (xray_structure extracted from PDB file) and reflection data. Fmodel has a broad array of methods that can imagine given data and model: R-factors calculation, bulk-solvent and scaling, refinement targets, and 100+ more.
- Compute initial R-factors (using model without DA added);
- Compute initial R-factors (using model plus DA);
- Start refinement. Loop over macro-cycles. Each macro-cycle may include (depending what's requested): occupancy, ADP, coordinate refinement (in the order as listed). Filtering and occupancy and B-factor values reset happens at each macro-cycle based on respective parameters.
- At the end, compute and output two maps: Fcalc map nased on DA only, and 2mFo-DFc using combined model (model + DA).

See code for details: ../mmtbx/grow_density.py