Appendix C. API Interface to the MPsim program.

When implementing a new driver it is important to have the following features correctly:

Various setups (degrees of freedom, non-bond and bond setups; energy expression.)

Call check_move and finish_move once in a while (see example below) to make sure that atoms that physically left cells to which they were assigned are reassigned to their new cells correctly.

(Note, usually it will be necessary to call the setup nonbonds immediately after that, to initialize the nonbond structures properly. Bond structures depend on absolute atom number, and not on which cell the atom is in, so it is not necessary to call bond setups)

Lastly, load balancing may be called once in a while, if executing on multiprocessors to redistribute the load more evenly.

Example of a main loop of a driver

```
calc_dof(); /* degrees of freedom (dof) goes into a global variable */
/* basic loop */
while (dyn_time < dyn_steps) {
   ...basic loop...
   setups if necessary.
   calc_ef_all - computes energy and forces.
   do_sync(). since calce_ef_all does not call it at the end.
   Perform moving (e.g. dynamics, minimization, something else)
   write trajectory entry...
   Call the functions that do load balancing and reassigning atoms between cells and cells between processors.
   do_sync(); 
   notmoving = (dyn_time + 1) % move_freq;
   check_move(!notmoving);
   if (!notmoving) finish_move();
   if (++local(loadbal_ctr) == loadbal_intvl) {
      load_balance();
      local(loadbal_ctr) = 0;
   }
   do_sync() ;
   write snap entry if desired.
```
if (here == 0) dyn_time ++ ;
  do_sync () ;
}

/* Note: can not use for, since dyn_time is shared */

---------------------------------------------------

Basic loop to process all atoms in parallel.

dynamic computation of global kinetic energy

float
calc_global_ke(void)
{
  int i ;
  do_sync();
  local(ke) = calc_cart_ke();/* each cpu will compute it’s ownKE */
  do_sync();
  if (here == 0) { /* node 0 also adds up the rigid KE */
    global_ke = calc_rigid_ke () ;
  }
  global_sum (global_ke,ke) ; /* node 0 will add them up */
  return global_ke ;
}

static float
calc_cart_ke(void)
{
  float ke, atom_ke;
  CELL *cell;
  ATOM *atom;
  ke = 0.0;

  init_next_cell();
  while ((cell = get_next_cell()) != NULL) {
    for (atom = cell->c_atom; atom != NULL; atom = atom->cnext)
    {
      /* this is the basic loop that goes over all the cells on this cpu and the over all the atoms contained in them */
      if (ATOM_NMOVE(atom)) continue;
      atom_ke = 0.0;
      atom_ke += atom->v[0] * atom->v[0];
      atom_ke += atom->v[1] * atom->v[1];
      ke += atom->m * atom_ke;
    }
  }
  return (ke * 0.5);
Loops in which atoms are processed in sequence can be done in the following way. This has to be done on one cpu only (usually 0; cpu)

```c
n = init_next_atom();
for (i = 0; i < n; i++) {
    ATOM *a = get_next_atom();
    ...
}
```

All the atom and cell interfaces are defined in atom.h and cell.h

ATOM a, can be tested with the following macro predicated for specific properties:

- `ATOM_FIXED(a)` true if atom is fixed.
- `ATOM_RIGID(a)` true if atom is part of a rigid molecule.
- `ATOM_NMOVE(a)` true if atom nonmovable, e.g. rigid or fixed.
- `ATOM_TRACK(a)` true if atom is being tracked.

Those properties can be set in the following way.

```c
a->flags |= ATOM_FIXED_FLAG;
```

The read_file biograf reader automatically sets the first one from data in the .bgf file and rigid and track are set from the corresponding keywords in the .ctl file.

`ATOM *get_atom(int gnum)` fetches an atom by absolute number.

```c
void add_force(ATOM *const a,
              const float fx, const float fy, const float fz,
              const char *const type);
```

/* this function adds force to the atom A. It can be used to implement new terms of the forcefield.

The string type, serves only debug-informational value.

All the cells interfaces are in cells.h

```c
void assign_cell(ATOM *atom); /* determines which cell this atom belongs to and assigns the value in atom->cell
void add_atom_to_cell(ATOM *atom); /* links the atom to its appropriate cell */
```