```
(** A mini version of ImpFS (imperative) *)

open Write_back_cache
type ('k,'v) wbc

type ('k,'v)map

type blk_id
type timestamp
(* avoid making an explict record for this *)
type ('k,'v) lru = (module Lru.F.S with type k='k and type v='v)

type did
type id
type dep

type op (* add, del *)

type dinode = {
  ptr_to_btree_root_and_pcache_root: unit (* FIXME *)
}

(** Type for the state of a particular directory; we use the
   convention that m is the in-memory state, and d is the on-disk
   state. *)
module Dir = struct
  (** The first field records whether the did has been synced to disk
     in the did_blk_map; once synced, it never changes; this allows us
     to avoid many potential syncs to the did_blk_map if we already
     know the entry is synced. *)
  type m = {
    did_synced_in_did_blk_map : bool;
    dinode_blk_id             : blk_id;
    dinode                    : dinode;
    entries                   : (string,op*dep option) wbc
  }
  type d = {
    entries_d       : (string,id)map;
    dinode_blk_id_d : blk_id;
    dinode_d        : dinode;
  }
end

(** This is to ensure that there is only one Dir.m for each did; we
   use "with_dir did" to take the relevant dir, and release it after
   updates *)
module Did_dir_map = struct
  type m = (did,Dir.m) map
end

(** A map from did to the blk_id for the corresponding dinode. Once
   created, entries are never modified; however, we need a sync
   operation to ensure that the did->blk_id entry is on disk before we
   link a new dir into its parent. We use "sync_1(did)" and record the
   sync status in the dir object (to avoid repeated unnecessary syncs)
   *)
module Did_blk_map = struct
  type m = {
    entries   : (did,blk_id)wbc;
  }
  type d = {
    entries_d: (did,blk_id)map;
  }
end
```