Help

```
#include "
href../common/optype_h_src.pdfoptype.h"
#include "
href../common/var_h_src.pdfvar.h"
#include "
href../common/ftools_h_src.pdftools.h"
#include "pnl/pnl_random.h"
#include "
href../common/error_msg_h_src.pdferror_msg.h"

int CHK_ok(int user, Planning *pt_plan, void *dum)
{
  return OK;
}

int CHK_call(int user, Planning *pt_plan, void *dum)
{
  NumFunc_1 *payoff = (NumFunc_1 *)dum;
  int status = OK;

  status += ChkParVar(pt_plan, payoff->Par);

  return status;
}

int CHK_callspread(int user, Planning *pt_plan, void *dum)
{
  NumFunc_1 *payoff = (NumFunc_1 *)dum;
  int status = OK;

  status += ChkParVar(pt_plan, payoff->Par);

  if (payoff->Par[1].Val.V_PDOUBLE < payoff->Par[0].Val.V_PDOUBLE)
    {
      Fprintf(TOSCREENANDFILE, "%s: lower than %s\ n", payoff->Par[1].Vname, pay
      status += 1;
    }

  return status;
```

```
}

int CHK_digit(int user, Planning *pt_plan, void *dum)
{
  NumFunc_1 *payoff = (NumFunc_1 *)dum;
  int status = OK;

  status += ChkParVar(pt_plan, payoff->Par);

  return status;
}

int CHK_tree(int user, Planning *pt_plan, void *dum)
{
  PricingMethod *Met = (PricingMethod *)dum;
  int status = OK;

  status += ChkParVar(pt_plan, Met->Par);

  return status;
}

int CHK_mc(int user, Planning *pt_plan, void *dum)
{
  PricingMethod *Met = (PricingMethod *)dum;
  int status = OK;

  status += ChkParVar(pt_plan, Met->Par);

  return status;
}

int CHK_mcBaldi(int user, Planning *pt_plan, void *dum)
{
  PricingMethod *Met = (PricingMethod *)dum;
  int status = OK;

  status += ChkParVar(pt_plan, Met->Par);

  return status;
}
```

```
int CHK_fdiff(int user, Planning *pt_plan, void *dum)
{
  PricingMethod *Met = (PricingMethod *)dum;
  int status = OK;

  status += ChkParVar(pt_plan, Met->Par);

  return status;
}

int CHK_split(int user, Planning *pt_plan, void *dum)
{
  PricingMethod *Met = (PricingMethod *)dum;
  int status = OK;

  status += ChkParVar(pt_plan, Met->Par);

  return status;
}

int CHK_psor(int user, Planning *pt_plan, void *dum)
{
  PricingMethod *Met = (PricingMethod *)dum;
  int status = OK;

  status += ChkParVar(pt_plan, Met->Par);

  return status;
}

extern int g_dup_printf;

/**
 * chk_mod_gen:
 * @param user:
 * @param pt_plan:
 * @param model: the model to be checked
 *
 * general model check function
 */
```

```
int chk_model_gen(int user, Planning *pt_plan, Model *model)
{
  void *pt = (model->TypeModel);
  int status = OK;
  int i, nvar = 0;
  VAR *var;
  char helpfile[MAX_PATH_LEN] = "";

  get_model_helpfile(model, helpfile);
  nvar = model->nvar;
  var = ((VAR *) pt);
  for (i = 0; i < nvar; i++)
    {
      status += ChkVar(pt_plan, &(var[i]));
    }
  return g_dup_printf ? status : Valid(user, status, helpfile);
}

/**
 * chk_opt_gen:
 * @param user:
 * @param pt_plan:
 * @param opt: the option to be checked
 *
 * general option check function
 */
int chk_option_gen(int user, Planning *pt_plan, Option *opt)
{
  void *pt = (opt->TypeOpt);
  int status = OK;
  int i, nvar = 0;
  VAR *var;
  char helpfile[MAX_PATH_LEN] = "";

  get_option_helpfile(opt, helpfile);

  nvar = opt->nvar;
  var = ((VAR *) pt);
  for (i = 0; i < nvar; i++)
    {
      if (var[i].Vsetable == SETABLE)
```

```
        switch (var[i].Vtype)
          {
          case  NUMFUNC_1:
            status += (var[i].Val.V_NUMFUNC_1)->Check(user, pt_plan, var[i].Val.
            break;
          case  NUMFUNC_2:
            status += (var[i].Val.V_NUMFUNC_2)->Check(user, pt_plan, var[i].Val.
            break;
          /* should be a type of FirstClass, check for
           * PtVar and PnlVect not implemented */
          default:
            status += ChkVar(pt_plan, &(var[i]));
            break;
          }
    }
  return g_dup_printf ? status : Valid(user, status, helpfile);
}
```