Help

```
#include "
href../../mod/copula/copula_h_src.pdfcopula.h"
#include "
href../../common/chk_h_src.pdfchk.h"
#include "
href../../mod/hes1d/hes1d_pad/model_h_src.pdfmodel.h"
#include "premia_obj.h"
#include "
href../../common/math/cdo/cdo_h_src.pdfmath/cdo/cdo.h"


static PremiaEnumMember CopulaTypeMembers[] =
{
  { "Gaussian", T_COPULA_GAUSS, 1},
  { "Clayton", T_COPULA_CLAYTON, 1},
  { "Normal Inverse Gaussian", T_COPULA_NIG, 1},
  { "Student", T_COPULA_STUDENT, 1},
  { "Double t", T_COPULA_DOUBLE_T, 1},
  { NULL, NULLINT, 0 }
};

static PremiaEnumMember IntensityTypeMembers[] =
{
  { "Non homogeneous",  0, 1},
  { "Homogeneous", 1, 1},
  { NULL, NULLINT, 0}
};

static DEFINE_ENUM(CopulaType, CopulaTypeMembers);
static DEFINE_ENUM(IntensityType, IntensityTypeMembers);

static PremiaEnumMember RecoveryTypeMembers[] =
{
  { "Constant", 1, 1 },
  { "Uniform", 2, 1 },
  { NULL, NULLINT, 0 }
};

static DEFINE_ENUM(RecoveryType, RecoveryTypeMembers);
```

```
/**
 * determine the number of parameters for the given type of copula
 *
 * @param n number of parameter (set on output)
 * @param t array containing on output the default parameters
 * @param copula_value an integer describing the type of copula
 * @param with_init if set to 1 t is initializes.
 * @return OK or WRONG
 */
static int n_param_copula(int *n, double *t, int copula_value, int with_init)
{
  if (with_init && t == NULL) return WRONG;
  switch (copula_value)
    {
    case T_COPULA_GAUSS:
      *n = 1;
      if (with_init)
        {
          t[0] = 0.03;
        }
      break;
    case T_COPULA_CLAYTON:
      *n = 1;
      if (with_init)
        {
          t[0] = 0.2;
        }
      break;
    case T_COPULA_NIG:
      *n = 3;
      if (with_init)
        {
          t[0] = 0.06;
          t[1] = 1.2558;
          t[2] = 0.2231;
        }
      break;
    case T_COPULA_STUDENT:
      *n = 2;
      if (with_init)
```

```
        {
          t[0] = 0.02;
          t[1] = 5;
        }
      break;
    case T_COPULA_DOUBLE_T:
      *n = 3;
      if (with_init)
        {
          t[0] = 0.03;
          t[1] = 5;
          t[2] = 7;
        }
      break;
    default:
      *n = 0;
      return WRONG;
      break;
    }
  return OK;
}

/**
 * determine the number of parameters for the given type of recovery
 *
 * @param n number of parameter (set on output)
 * @param t array containing on output the default parameters
 * @param recovery_value an integer describing the type of recovery
 * @param with_init if set to 1 t is initializes.
 * @return OK or WRONG
 */
static int n_param_recovery(int *n, double *t, int recovery_value, int with_init
{
  if (with_init && t == NULL) return WRONG;
  switch (recovery_value)
    {
    /* Constant */
    case 1:
      *n = 1;
      if (with_init)
        {
```

```
          t[0] = 0.4;
        }
      break;
    /* Uniform */
    case 2:
      *n = 2;
      if (with_init)
        {
          t[0] = 0.3;
          t[1] = 0.5;
        }
      break;
    default:
      *n = 0;
      return WRONG;
      break;
    }
  return OK;
}


/**
 * Initialization of the Copula Model
 * @param model
 */
static int MOD(Init)(Model *model)
{
  VAR *Par;
  double t[3];
  int n_copula, n_recovery;
  PremiaEnumMember *e;
  TYPEMOD *pt = (TYPEMOD *)(model->TypeModel);
  if (model->init == 0)
    {
      model->init = 1;
      model->nvar = 0;
      pt->Ncomp.Vname = "Number of Companies";
      pt->Ncomp.Vtype = PINT;
      pt->Ncomp.Val.V_PINT = 100;
      pt->Ncomp.Viter = ALLOW;
      model->nvar++;
```

```
pt->r.Vname = "Interest rate";
pt->r.Vtype = PDOUBLE;
pt->r.Val.V_PDOUBLE = 0.04;
pt->r.Viter = ALLOW;
model->nvar++;

pt->t_copula.Vname = "Copula";
pt->t_copula.Vtype = ENUM;
pt->t_copula.Val.V_ENUM.value = 1;
pt->t_copula.Val.V_ENUM.members = &CopulaType;
pt->t_copula.Viter = FORBID;
model->nvar++;

for (e = pt->t_copula.Val.V_ENUM.members->members ; e->label != NULL ; e++
  {
    e->nvar = 1;
    e->Par[0].Vname = "Copula Parameters";
    e->Par[0].Viter = FORBID;
    e->Par[0].Vtype = PNLVECT;
    if (n_param_copula(&n_copula, t,  e->key, 1) != OK) return WRONG;
    e->Par[0].Val.V_PNLVECT = pnl_vect_create_from_ptr(n_copula, t);
  }

pt->t_intensity.Vname = "Homogeneous Intensity";
pt->t_intensity.Vtype = ENUM;
pt->t_intensity.Val.V_ENUM.value = 1;
pt->t_intensity.Val.V_ENUM.members = &IntensityType;
pt->t_intensity.Viter = FORBID;
model->nvar++;

e = &(pt->t_intensity.Val.V_ENUM.members->members[0]);
e->nvar = 1;
e->Par[0].Vname = "Intensity";
e->Par[0].Vtype = FILENAME;
e->Par[0].Val.V_FILENAME = NULL;
e->Par[0].Viter = FORBID;
e = &(pt->t_intensity.Val.V_ENUM.members->members[1]);
e->nvar = 1;
e->Par[0].Vname = "Intensity";
e->Par[0].Vtype = PDOUBLE;
```

```
    e->Par[0].Val.V_PDOUBLE = 0.01;
    e->Par[0].Viter = FORBID;

    pt->t_recovery.Vname = "Recovery type";
    pt->t_recovery.Vtype = ENUM;
    pt->t_recovery.Val.V_ENUM.value = 1;
    pt->t_recovery.Val.V_ENUM.members = &RecoveryType;
    pt->t_recovery.Viter = FORBID;
    model->nvar++;

    Par = lookup_premia_enum_par(&(pt->t_recovery), 1);
    Par[0].Viter = FORBID;
    Par[0].Vsetable = SETABLE;
    Par[0].Vtype = DOUBLE;
    Par[0].Val.V_DOUBLE = 0.4;
    Par[0].Vname = "Recovery";

    Par = lookup_premia_enum_par(&(pt->t_recovery), 2);
    Par[0].Viter = FORBID;
    Par[0].Vsetable = SETABLE;
    Par[0].Vtype = PNLVECT;
    Par[0].Val.V_PNLVECT = NULL;
    Par[0].Vname = "Recovery";

  }

/* Recovery vector */
Par = lookup_premia_enum_par(&(pt->t_recovery), 2);
if (Par[0].Val.V_PNLVECT == NULL)
  {
    if (n_param_recovery(&n_recovery, t,  2, 1) != OK) return WRONG;
    Par[0].Val.V_PNLVECT = pnl_vect_create_from_ptr(n_recovery, t);
  }


e = &(pt->t_intensity.Val.V_ENUM.members->members[0]);
if (e->Par[0].Val.V_FILENAME == NULL)
  {
    if ((e->Par[0].Val.V_FILENAME = malloc(sizeof(char) * MAX_PATH_LEN)) == NU
      return MEMORY_ALLOCATION_FAILURE;
    sprintf(e->Par[0].Val.V_FILENAME, "%s%scdo_intensity.dat", premia_data_dir
```

```
    }
  return OK;
}

TYPEMOD Copula;

MAKEMOD(Copula);
```