

Help

```

#include "optype.h"
#include "var.h"
#include "tools.h"
#include "pnl/pnl_random.h"
#include "error_msg.h"

/**
 * Get_model:
 * @param user:
 * @param pt_plan:
 * @param model:
 *
 * generic function to interactively read the model
 * parameters
 */
int Get_model_gen(int user, Planning *pt_plan, Model *model)
{
    int nvar;
    void *pt = (model->TypeModel);
    VAR *var = ((VAR *) pt);
    int i;

    model->Init(model);
    nvar = model->nvar;
    if (user == TOSCREEN)
        if ((model->Show)(user, pt_plan, model))
            do
            {
                Fprintf(TOSCREEN, "_____Model:%s\ n", model->N
                    for (i = 0; i < nvar; i++)
                    {
                        ScanVar(pt_plan, user, &(var[i]));
                        if (var[i].setter) var[i].setter(model->TypeModel);
                    }
            }
            while ((model->Show)(user, pt_plan, model));

```

```

    return ((model->Show)(TOSCREENANDFILE, pt_plan, model));
}

/**
 * FGet_model
 * @param InputFile:
 * @param user:
 * @param pt_plan:
 * @param model:
 *
 * generic function to read the model parameters from an
 * input file
 */
int FGet_model_gen(char **InputFile, int user, Planning *pt_plan, Model *model)
{
    int nvar;
    void *pt = (model->TypeModel);
    VAR *var = ((VAR *) pt);
    int i;

    model->Init(model);
    nvar = model->nvar;
    if (user == TOSCREEN)
        Fprintf(TOSCREEN, "-----Model:%s\ n", model->Name);

    for (i = 0; i < nvar; i++)
    {
        FScanVar(InputFile, pt_plan, user, &(amp;var[i]));
        if (var[i].setter) var[i].setter(model->TypeModel);
    }
    return ((model->Show)(TOSCREENANDFILE, pt_plan, model));
}

/**
 * Generic function to replace the Show member function of
 * the model structures
 *
 * @param user : an integer TOSCREEN or TOFILE
 * @param pt_plan : pointer the planning structure

```

```

* describing what to do
* @param model : pointer to the model instance
*/
int Show_model_gen(int user, Planning *pt_plan, Model *model)
{
    void *pt = (model->TypeModel);
    VAR *var = ((VAR *)pt);
    VAR _bs;
    int nvar = model->nvar, i, j;
    char *id = NULL;

    Fprintf(user, "##Model:%s\ n", model->Name);

    for (i = 0; i < nvar; i++)
    {
        PrintVar(pt_plan, user, &(var[i]));
        if (strcmp(var[i].Vname, "Annual Interest Rate", 20) == 0 ||
            strcmp(var[i].Vname, "Annual Dividend Rate", 20) == 0)
        {
            if (id == NULL)
            {
                if ((id = malloc(33 * sizeof(char))) == NULL)
                    return MEMORY_ALLOCATION_FAILURE;
            }
            _bs.Vtype = DOUBLE;
            _bs.Val.V_DOUBLE = log(1 + var[i].Val.V_DOUBLE / 100);
            strcpy(id, "-->Instantaneous");
            strcat(id, &(var[i].Vname[6]));
            _bs.Vname = id;
            _bs.Viter = FORBID;
            _bs.setter = NULL;
            _bs.Vsetable = SETABLE;
            if (var[i].Vtype == PNLVECT)
            {
                strcat(id, ": ");
                Fprintf(user, id);
                for (j = 0; j < var[i].Val.V_PNLVECT->size; j++)
                {
                    _bs.Val.V_DOUBLE = log(1 + var[i].Val.V_PNLVECT->array[j] / 10
                    Fprintf(user, "%f ", _bs.Val.V_DOUBLE);
                }
            }
        }
    }
}

```

```
        Fprintf(user, "\ n");
    }
    else
    {
        _bs.Val.V_DOUBLE = log(1 + var[i].Val.V_DOUBLE / 100);
        PrintVar(pt_plan, user, &_bs);
    }
}
}
if (id != NULL)
{
    free(id);
    id = NULL;
}
return (model->Check)(user, pt_plan, model);
}
```