

[Help](#)

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

extern char premia_man_dir[MAX_PATH_LEN];

#if (defined _WIN32 || defined _MSC_VER)
char *rindex(const char *s, int c)
{
    const char *last = NULL;

    while (*s != '\0')
    {
        if (*s == c) last = s;
        s++;
    }
    return (char *)last;
}
#else
#include <strings.h>
#endif /* (defined _WIN32 || defined _MSC_VER) */

/*-----PRICINGMETHODS-----*/
int FGetMethod(char **InputFile, int user, Planning *pt_plan, Pricing *Pr, Prici
{

    (Met->Init)(Met, opt);
    if (user == TOSCREEN)
    {
```

```

        Fprintf(TOSCREEN, "%s\ n", Met->Name);
        if (pt_plan->NumberOfMethods != 0)
            FixMethod(pt_plan, Met);
        FGetParVar(InputFile, pt_plan, user, Met->Par);
    }
    return ShowMethod(TOSCREENANDFILE, pt_plan, Pr, Met, opt);
}

int GetMethod(int user, Planning *pt_plan, Pricing *Pr, PricingMethod *Met, Opti
{
    (Met->Init)(Met, opt);
    if (user == TOSCREEN)
    {
        if (ShowMethod(user, pt_plan, Pr, Met, opt))
        {
            do
            {
                Fprintf(TOSCREEN, "%s\ n", Met->Name);
                if (pt_plan->NumberOfMethods != 0)
                    FixMethod(pt_plan, Met);
                GetParVar(pt_plan, user, Met->Par);
            }
            while (ShowMethod(user, pt_plan, Pr, Met, opt));
        }
    }
    return ShowMethod(TOSCREENANDFILE, pt_plan, Pr, Met, opt);
}

int ShowMethod(int user, Planning *pt_plan, Pricing *Pr, PricingMethod *Met, Opti
{
    char helpfile[MAX_PATH_LEN] = "";

    (Met->Init)(Met, opt);
    get_method_helpfile(Pr, Met, helpfile);

    if (user == TOSCREEN)
        FixMethod(pt_plan, Met);

    if (user == TOSCREENANDFILE)
        ShowParVar(pt_plan, user, Met->Par);

```

```

else
{
    if (ShowParVar(pt_plan, user, Met->Par) == OK)
        return Valid(user, (Met->Check)(user, pt_plan, Met) + ChkParVar(pt_plan,
    else
        return Valid(NO_PAR, ChkParVar(pt_plan, Met->Par), helpfile);
}
return OK;
}

int ShowResultMethod(int user, Planning *pt_plan, int error, PricingMethod *Met)
{
    if ((error == 0) || (user == NAMEONLYTOFILE))
    {
        ShowParVar(pt_plan, user, Met->Res);
    }
    else
    {
        Fprintf(user, "%s\ n", error_msg[error]);
    }
    return OK;
}

int FixMethod(Planning *pt_plan, PricingMethod *pt_method)
{
    int i, j;
    char msg, answer;

    if (pt_plan->NumberOfMethods == 0)
        return OK;

    for (j = 0; j < pt_plan->VarNumber && pt_plan->Par[j].Location->Vtype != PREMI
    {
        for (i = 0; pt_method->Par[i].Vtype != PREMIA_NULLTYPE; ++i)
        {
            if (!strcmp(pt_method->Par[i].Vname, pt_plan->Par[j].Location->Vname))
            {
                pt_plan->Par[j].Location = &pt_method->Par[i];
                pt_method->Par[i].Viter = ALREADYITERATED + j;
                return DONOTITERATE;
            }
        }
    }
}

```

```

else if (CompareParameterNames(pt_method->Par[i].Vname, pt_plan->Par[j]
{
    Fprintf(TOSCREEN, "\ nWould you like to iterate \ \"%s\ " like \ \"%
    answer = (char)tolower(fgetc(stdin));
    msg = answer;
    while ((answer != '\ n') && (answer != EOF))
        answer = (char)fgetc(stdin);
    if (msg == '\ n')
    {
        pt_plan->Par[j].Location = &pt_method->Par[i];
        pt_method->Par[i].Viter = ALREADYITERATED + j;
        return DONOTITERATE;
    }
}
else if (pt_method->Par[i].Vtype == pt_plan->Par[j].Location->Vtype)
{
    Fprintf(TOSCREEN, "\ nWould you like to iterate \ \"%s\ " like \ \"%
    answer = (char)tolower(fgetc(stdin));
    msg = answer;
    while ((answer != '\ n') && (answer != EOF))
        answer = (char)fgetc(stdin);
    if (msg == '\ n')
    {
        pt_plan->Par[j].Location = &pt_method->Par[i];
        pt_method->Par[i].Viter = ALREADYITERATED + j;
        return DONOTITERATE;
    }
}
}
}
return OK;
}

/**
 * This functions compares 2 strings searching for the occurence of one within
 *
 * @param s1
 * @param s2
 *
 * @return OK or WRONG (OK if either s1 is contained in s2 or s2 is contained i
 * All leading '#' are stripped out

```

```

*/
int CompareParameterNames(const char *s1, const char *s2) /* */
{
    int len, i;
    const char *small, *large;
    if (strlen(s1) <= strlen(s2))
    {
        small = s1;
        large = s2;
    }
    else
    {
        small = s2;
        large = s1;
    }
    /* strip away beginning '#' characters */

    if (small[0] == '#') ++small;
    if (large[0] == '#') ++large;
    len = strlen(large);
    for (i = 0; i <= (int)(len - strlen(small)); ++i)
    {
        if (strncmp(small, large, strlen(small)) == 0) return OK;
        ++large;
    }
    return WRONG;
}

```