

[Help](#)

```
#include "hullwhite1dgeneralized_std.h"

int MOD_OPT(ChkMix)(Option *Opt, Model *Mod)
{
    TYPEOPT *ptOpt = (TYPEOPT *) (Opt->TypeOpt);
    TYPEMOD *ptMod = (TYPEMOD *) (Mod->TypeModel);
    int status = OK;

    if ((strcmp(Opt->Name, "ZeroCouponCallBondEuro") == 0) || (strcmp(Opt->Name, "
    {
        if ((ptOpt->OMaturity.Val.V_DATE) <= (ptMod->T.Val.V_DATE))
        {
            Fprintf(TOSCREENANDFILE, "Current date greater than maturity!\ n");
            status += 1;
        }
        if ((ptOpt->BMaturity.Val.V_DATE) <= (ptOpt->OMaturity.Val.V_DATE))
        {
            Fprintf(TOSCREENANDFILE, "Option maturity greater than Bond maturity!\ n
            status += 1;
        }
    }
    if ((strcmp(Opt->Name, "ZeroCouponBond") == 0))
    {
        if ((ptOpt->BMaturity.Val.V_DATE) <= (ptMod->T.Val.V_DATE))
        {
            Fprintf(TOSCREENANDFILE, "Current date greater than maturity!\ n");
            status += 1;
        }
    }
    if ((strcmp(Opt->Name, "PayerSwaption") == 0) || (strcmp(Opt->Name, "ReceiverS
        if ((ptOpt->BMaturity.Val.V_DATE) <= (ptOpt->OMaturity.Val.V_DATE))
        {
            Fprintf(TOSCREENANDFILE, "Option maturity greater than Bond maturity!\ n
            status += 1;
        }
    }

    if ((strcmp(Opt->Name, "Floor") == 0) || (strcmp(Opt->Name, "Cap") == 0))
    {
        if ((ptOpt->FirstResetDate.Val.V_DATE) <= (ptMod->T.Val.V_DATE))
```

```

        {
            Fprintf(TOSCREENANDFILE, "Current date greater than first coupon date!");
            status += 1;
        }
    if ((ptOpt->FirstResetDate.Val.V_DATE) >= (ptOpt->BMaturity.Val.V_DATE))
    {
        Fprintf(TOSCREENANDFILE, "First reset date greater than contract matur");
        status += 1;
    }
}

return status;
}

extern PricingMethod MET(CF_ZCCallBondEuroHW1DG);
extern PricingMethod MET(CF_ZCPutBondEuroHW1DG);
extern PricingMethod MET(CF_CapHW1dG);
extern PricingMethod MET(CF_FloorHW1dG);
extern PricingMethod MET(CF_PayerSwaptionHW1dG);
extern PricingMethod MET(CF_ReceiverSwaptionHW1dG);
extern PricingMethod MET(TR_ZCBondHW1dG);
extern PricingMethod MET(TR_ZBOHW1dG);
extern PricingMethod MET(TR_CapFloorHW1dG);
extern PricingMethod MET(TR_SwaptionHW1dG);
extern PricingMethod MET(TR_BermudianSwaptionHW1DG);

PricingMethod *MOD_OPT(methods) [] =
{
    &MET(CF_ZCCallBondEuroHW1DG),
    &MET(CF_ZCPutBondEuroHW1DG),
    &MET(CF_CapHW1dG),
    &MET(CF_FloorHW1dG),
    &MET(CF_PayerSwaptionHW1dG),
    &MET(CF_ReceiverSwaptionHW1dG),
    &MET(TR_ZCBondHW1dG),
    &MET(TR_ZBOHW1dG),
    &MET(TR_CapFloorHW1dG),
    &MET(TR_SwaptionHW1dG),
    &MET(TR_BermudianSwaptionHW1DG),

```

```
    NULL
};
DynamicTest *MOD_OPT(tests) [] =
{
    NULL
};
```

```
Pricing MOD_OPT(pricing) =
{
    ID_MOD_OPT,
    MOD_OPT(methods),
    MOD_OPT(tests),
    MOD_OPT(ChkMix)
};
```