Help

```
#include <stdlib.h>
#include  "hes1d_std.h"
#include "pnl/pnl_basis.h"
#include  "math/alfonsi.h"
#include "enums.h"

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion <
     (2011+2) //The "#else" part of the code will be freely av
    ailable after the (year of creation of this file + 2)
static int CHK_OPT(MC_AM_Alfonsi_Iterative)(void *Opt, voi
    d *Mod)
{
    return NONACTIVE;
}
int CALC(MC_AM_Alfonsi_Iterative)(void *Opt, void *Mod,
    PricingMethod *Met)
{
    return AVAILABLE_IN_FULL_PREMIA;
}
#else

#define nbr_var_explicatives_LoSc 2


/** The principle the iterative algorithm is start with a
    first exercise strategy (verifying some properties) and cons
    truct a second one that will be (theoretically) closer to th
    e optimal strategy.
In this code we consider two different initial strategies:

1) The strategy given by the Longstaff-Schwartz algorithm
2) The strategy that exercises when the payoff is greater
    than the prices of all the lasting European options.
**/

/** Estimate the exercise strategy given by the Longstaff-
    Schwartz algorithm **/
/* Exercice dates are : T(0), T(1), ..., T(NbrExerciseDate
    s-1), with T(0)=0 and T(NbrExerciseDates-1)=Maturity.
The exercise strategy is: at time T(i) we exercise if Dis
```

```
    countedPayoff(T(i))>=Continuationvalue(T(i))
where Continuationvalue(T(i)) is given by regession. The
    regession coefficient are stored in the matrix RegressionCoe
    ffMat*/
static int MC_Am_Alfonsi_LoSc(NumFunc_1 *p, double S0,
    double Maturity, double r, double divid, double V0, double kapp
    a, double theta, double sigma, double rho, int basis_name,
    int DimApprox, PnlMat *SpotPaths, PnlMat *VarPaths, PnlMat*
    RegressionCoeffMat)
{
    int j, m;
    double regressed_value, discounted_payoff, S_t, V_t,
    discount, discount_step, time_step, exercise_date, Continua
    tionValue_0;
    double *VariablesExplicatives;
    int NbrMCsimulation=SpotPaths->n, NbrExerciseDates=Spo
    tPaths->m;

    PnlMat *ExplicativeVariables;
    PnlVect *DiscountedOptimalPayoff, *RegressionCoeffVect;
    PnlBasis *basis;

    pnl_mat_resize(RegressionCoeffMat, NbrExerciseDates,
    DimApprox);
    pnl_mat_set_double(RegressionCoeffMat, 0.);

    time_step = Maturity / (NbrExerciseDates-1);
    discount_step = exp(-r*time_step);
    discount = exp(-r*Maturity);

    /* We store Spot and Variance*/

    basis = pnl_basis_create(basis_name, DimApprox, nbr_
    var_explicatives_LoSc);

    VariablesExplicatives = malloc(nbr_var_explicatives_LoS
    c*sizeof(double));

    ExplicativeVariables = pnl_mat_create(NbrMCsimulation,
    nbr_var_explicatives_LoSc);
    DiscountedOptimalPayoff = pnl_vect_create(NbrMCsimulat
```

```
ion); // Continuation Value

RegressionCoeffVect = pnl_vect_create(0);

// At maturity, the price of the option = discounted_
payoff
exercise_date = Maturity;
for (m=0; m<NbrMCsimulation; m++)
{
    S_t = MGET(SpotPaths, NbrExerciseDates-1, m);
    LET(DiscountedOptimalPayoff, m) = discount * (p->
Compute)(p->Par, S_t); // Price of the option = discounted_
payoff
}

// Backward iterations
for (j=NbrExerciseDates-2; j>=1; j--)
{
    /** Least square fitting **/
    exercise_date -= time_step;
    discount /= discount_step;

    for (m=0; m<NbrMCsimulation; m++)
    {
        V_t = MGET(VarPaths, j, m); // Simulated value
of the variance
        S_t = MGET(SpotPaths, j, m); // Simulated value
 of the spot

        MLET(ExplicativeVariables, m, 0) = S_t/S0;
        MLET(ExplicativeVariables, m, 1) = V_t/V0;
    }

    pnl_basis_fit_ls(basis, RegressionCoeffVect, Explic
ativeVariables, DiscountedOptimalPayoff);

    // Save regression coefficients in RegressionCoeffM
at.
    pnl_mat_set_row(RegressionCoeffMat, RegressionCoe
ffVect, j);
```

```
    /** Dynamical programming equation **/
    for (m=0; m<NbrMCsimulation; m++)
    {
        V_t = MGET(VarPaths, j, m);
        S_t = MGET(SpotPaths, j, m);
        discounted_payoff = discount * (p->Compute)(p->
Par, S_t);

        if (discounted_payoff>0) // If the discounted_
payoff is null, the OptimalPayoff doesnt change.
        {
            VariablesExplicatives[0] = S_t/S0;
            VariablesExplicatives[1] = V_t/V0;

            regressed_value = pnl_basis_eval(basis,Reg
ressionCoeffVect, VariablesExplicatives);

            if (discounted_payoff > regressed_value)
            {
                LET(DiscountedOptimalPayoff, m) = dis
counted_payoff;
            }
        }
    }
}

// At initial date, no need for regression, cond.expec
tation is just a plain expectation, estimated with empiric
al mean.
ContinuationValue_0 = pnl_vect_sum(DiscountedOptimalPay
off)/NbrMCsimulation;

MLET(RegressionCoeffMat, 0, 0) = ContinuationValue_0;

free(VariablesExplicatives);
pnl_basis_free (&basis);
pnl_mat_free(&ExplicativeVariables);

pnl_vect_free(&DiscountedOptimalPayoff);
pnl_vect_free(&RegressionCoeffVect);
```

```
    return OK;
}


// Initial strategy given by Longstaff-Schwartz algorithm
double LimInitialStrategy_LoSc(NumFunc_1 *p, int i, int Nb
    rExerciseDates, double time_step, double S_i, double S0,
    double discount_i, double r, double divid, double V_i, double V0
    , double kappa, double theta, double sigma, double rho, Pn
    lBasis *basis, PnlVect *RegCoeffVect_LoSc)
{
    double result, *VariablesExplicatives;

    VariablesExplicatives = malloc(nbr_var_explicatives_LoS
    c*sizeof(double));

    VariablesExplicatives[0] = S_i/S0;
    VariablesExplicatives[1] = V_i/V0;

    result = MAX(0., pnl_basis_eval(basis, RegCoeffVect_LoS
    c, VariablesExplicatives));
    free(VariablesExplicatives);

    return result;
}

// With this initial strategy, we exercise if the payoff is
     greater than the maximum of lasting european options
double LimInitialStrategy_EuOpt(NumFunc_1 *p, int i, int Nb
    rExerciseDates, double time_step, double S_i, double discoun
    t_i, double r, double divid, double V_i, double kappa,
    double theta, double sigma, double rho)
{
    double Q_0_tilde_i, european_price=0., european_delta=0
    .;
    int q;

    if (i==NbrExerciseDates-1) return 0.;
    else
    {
        Q_0_tilde_i =  discount_i * (p->Compute)(p->Par, S_
```

```
    i);

        for (q=i+1; q<NbrExerciseDates; q++)
        {
            ApAntonelliScarlattiHeston(S_i, p, (q-i)*time_s
    tep, r, divid, V_i, kappa, theta, sigma, rho, &european_
    price, &european_delta);
            Q_0_tilde_i = MAX(Q_0_tilde_i, discount_i*euro
    pean_price);
        }

        return Q_0_tilde_i;
    }
}

/* Initial Strategy that will be improved with iterative
    algorithm.
This initial strategy is defined by:
We exercise at t(i) if DiscountedPayoff(t(i)) is greater th
    an LimInitialStrategy(...) */
double LimInitialStrategy(NumFunc_1 *p, int i, int NbrExerc
    iseDates, double time_step, double S_i, double S0, double
    discount_i,
                        double r, double divid, double V_
    i, double V0, double kappa, double theta, double sigma,
    double rho,
                        PnlBasis *basis, PnlVect *RegCoe
    ffVect_LoSc, int flag_InitStrategy)
{
    double Q_0_tilde_i=0.;

    if (flag_InitStrategy==1)
    {
        Q_0_tilde_i = LimInitialStrategy_EuOpt(p, i, NbrEx
    erciseDates, time_step, S_i, discount_i, r, divid, V_i, kapp
    a, theta, sigma, rho);
    }

    else if (flag_InitStrategy==2)
    {
        Q_0_tilde_i = LimInitialStrategy_LoSc(p, i, NbrEx
```

```
    erciseDates, time_step, S_i, S0, discount_i,r, divid, V_i,
    V0, kappa, theta, sigma, rho, basis, RegCoeffVect_LoSc);
    }

    return Q_0_tilde_i;
}

/** Price of american put/call option using Iterative algor
    ithm by Kolodko et al.**/
/** Heston model is simulated using the method proposed by
    Alfonsi **/
// Exercice dates are : T(0), T(1), ..., T(NbrExerciseDate
    s-1).
// with T(0)=0 and T(NbrExerciseDates-1)=Maturity.
static int MC_Am_Alfonsi_Iterative(NumFunc_1 *p, double S0,
     double Maturity, double r, double divid, double V0,
    double kappa, double theta, double sigma, double rho, int flag_
    InitStrategy, long NbrMCsimulation, int NbrExerciseDates,
    int NbrStepPerPeriod, int generator, int flag_cir, int basis_
    name, int DimApprox, int WindowPar, double *ptPriceAm_2nd
    Strategy, double *ptPriceAm_1stStrategy)
{
    int i, j, m, q, init_mc;
    int flag_SpotPaths, flag_VarPaths, flag_AveragePaths;
    double discounted_payoff_j, discounted_payoff_i=0., dis
    count_step, time_step;
    double S_j, V_j, Q_0_tilde_i=0., exercise_date, S_i, V_
    i, regressed_value;
    double *VariablesExplicatives;

    PnlVect *discount, *RegCoeffVect_LoSc, *RegCoeffVect_
    Iter, *DiscountedOptimalPayoff, *VectToReg;
    PnlMat *SpotPaths, *VarPaths, *AveragePaths,
    *RegCoeffMat_LoSc, *RegCoeffMat_Iter, *ExplicativeVaria
    bles, *CondExp;

    PnlMatInt *ExerciseDecision;

    PnlBasis *basis;

    init_mc=pnl_rand_init(generator, NbrExerciseDates*Nb
```

```
rStepPerPeriod, NbrMCsimulation);
if (init_mc != OK) return init_mc;

/* We store Spot and Variance*/
flag_SpotPaths = 1;
flag_VarPaths  = 1;
flag_AveragePaths = 0;

SpotPaths = pnl_mat_new(); // Matrix of the whole traj
ectories of the spot
VarPaths = pnl_mat_new(); // Matrix of the whole trajec
tories of the variance
AveragePaths = pnl_mat_new(); // This variable wont be
used
RegCoeffVect_LoSc = pnl_vect_new(); // Regression coe
fficients given by Longstaff-Schwartz algorithm
RegCoeffMat_LoSc = pnl_mat_new(); // All Regression coe
fficients given by Longstaff-Schwartz algorithm
RegCoeffVect_Iter = pnl_vect_new(); // Regression coe
fficients in the Iterative Algorithm
RegCoeffMat_Iter = pnl_mat_create(NbrExerciseDates, Dim
Approx); // All Regression coefficients in the Iterative Alg
orithm
DiscountedOptimalPayoff = pnl_vect_create(NbrMCsimulat
ion); // Value of the Discounted Payoff at the stopping time
VectToReg = pnl_vect_create(NbrMCsimulation);
ExplicativeVariables = pnl_mat_create(NbrMCsimulation,
nbr_var_explicatives_LoSc);

//ExerciseDecision[i,m]=0 the strategy says "No exercis
e" at time t(i) for simulation m.
//ExerciseDecision[i,m]=1 the strategy says "Exercise"
at time t(i) for simulation m.
ExerciseDecision = pnl_mat_int_create(NbrExerciseDates,
 NbrMCsimulation);
pnl_mat_int_set_int(ExerciseDecision, 0);

VariablesExplicatives = malloc(nbr_var_explicatives_LoS
c*sizeof(double));

CondExp = pnl_mat_create(NbrExerciseDates, NbrMCsimulat
```

```
ion);
discount = pnl_vect_create(NbrExerciseDates);

time_step = Maturity / (double)(NbrExerciseDates-1);
discount_step = exp(-r*time_step);
LET(discount, 0) = 1.;
for (i=1; i<NbrExerciseDates; i++) LET(discount, i) =
discount_step*GET(discount, i-1);

basis = pnl_basis_create(basis_name, DimApprox, nbr_
var_explicatives_LoSc);

// Simulation of the whole paths
HestonSimulation_Alfonsi(flag_SpotPaths, SpotPaths, fla
g_VarPaths, VarPaths, flag_AveragePaths, AveragePaths, S0,
Maturity, r, divid, V0, kappa, theta, sigma, rho, NbrMCsimu
lation, NbrExerciseDates, NbrStepPerPeriod, generator, flag_
cir);

// If flag_InitStrategy=2, the first strategy will the
Longstaff-Schwartz one
if (flag_InitStrategy==2) MC_Am_Alfonsi_LoSc(p, S0,
Maturity, r, divid, V0, kappa, theta, sigma, rho, basis_name,
 DimApprox, SpotPaths, VarPaths, RegCoeffMat_LoSc);


///* Here we compute the dates where the initial stra
tegy says to exercise the option. So:
//ExerciseDecision[i,m]=0 the strategy says "No exercis
e" at time t(i) for simulation m.
//ExerciseDecision[i,m]=1 the strategy says "Exercise"
at time t(i) for simulation m.
i=0; // t=0
S_i = S0;
V_i = V0;
discounted_payoff_i = GET(discount, i) * (p->Compute)(
p->Par, S_i);
if (discounted_payoff_i>0)
{
    if (flag_InitStrategy==2) pnl_mat_get_row(RegCoeffV
ect_LoSc, RegCoeffMat_LoSc, i);
```

```
    Q_0_tilde_i = LimInitialStrategy(p, i, NbrExerciseD
ates, time_step, S_i, S0, GET(discount, i), r, divid, V_i,
V0, kappa, theta, sigma, rho, basis, RegCoeffVect_LoSc, fla
g_InitStrategy);

    if (discounted_payoff_i >= Q_0_tilde_i)
    {
        for (m=0; m<NbrMCsimulation; m++)
            pnl_mat_int_set(ExerciseDecision, i, m, 1);
    }
}

for (i=1; i<NbrExerciseDates-1; i++)
{
    if (flag_InitStrategy==2)
        pnl_mat_get_row(RegCoeffVect_LoSc, RegCoeffMat_
LoSc, i);

    for (m=0; m<NbrMCsimulation; m++)
    {
        S_i = MGET(SpotPaths, i, m);
        V_i = MGET(VarPaths, i, m);

        discounted_payoff_i = GET(discount, i) * (p->
Compute)(p->Par, S_i);

        if (discounted_payoff_i>0)
        {
            Q_0_tilde_i = LimInitialStrategy(p, i, Nb
rExerciseDates, time_step, S_i, S0, GET(discount, i), r, div
id, V_i, V0, kappa, theta, sigma, rho, basis, RegCoeffVect_
LoSc, flag_InitStrategy);

            if (discounted_payoff_i >= Q_0_tilde_i)
                pnl_mat_int_set(ExerciseDecision, i, m,
 1);
        }
    }
}
```

```
for (m=0; m<NbrMCsimulation; m++)
    pnl_mat_int_set(ExerciseDecision, NbrExerciseDates-
1, m, 1);


///* We compute the option price with the initial exerc
ise strategy
*ptPriceAm_1stStrategy=0.;
for (m=0; m<NbrMCsimulation; m++)
{
    i=-1;
    do
    {
        i++;
    }
    while (pnl_mat_int_get(ExerciseDecision, i, m)==0);

    S_i = MGET(SpotPaths, i, m);
    V_i = MGET(VarPaths, i, m);
    discounted_payoff_i = GET(discount, i) * (p->Compu
te)(p->Par, S_i);
    *ptPriceAm_1stStrategy += discounted_payoff_i;
}
*ptPriceAm_1stStrategy /= (double) NbrMCsimulation; //
Price with initial exercise strategy


///* Now we compute the option price following the impr
oved exercise strategy
// At maturity, the price of the option = discounted_
payoff
// DiscountedOptimalPayoff will contain the disc.payo
ff following the improved exercise strategy
exercise_date = Maturity;
for (m=0; m<NbrMCsimulation; m++)
{
    S_j = MGET(SpotPaths, NbrExerciseDates-1, m);
    LET(DiscountedOptimalPayoff, m) = GET(discount, Nb
rExerciseDates-1) * (p->Compute)(p->Par, S_j);
}
```

```
for (j=NbrExerciseDates-2; j>=1; j--)
{
    exercise_date -= time_step;

    for (m=0; m<NbrMCsimulation; m++)
    {
        S_j = MGET(SpotPaths, j, m);
        V_j = MGET(VarPaths, j, m);

        MLET(ExplicativeVariables, m, 0) = S_j/S0;
        MLET(ExplicativeVariables, m, 1) = V_j/V0;

        i = j-1;
        for (q=j; q<MIN(j+WindowPar+1, NbrExerciseDate
s); q++)
        {
            if(i<q)
            {

                do
                {
                    i++;
                }
                while (pnl_mat_int_get(ExerciseDecisio
n, i, m)==0);

                S_i = MGET(SpotPaths, i, m);
                V_i = MGET(VarPaths, i, m);
                discounted_payoff_i = GET(discount, i)
* (p->Compute)(p->Par, S_i);

                MLET(CondExp, q, m) = discounted_payo
ff_i;
            }
            else MLET(CondExp, q, m) = discounted_payo
ff_i;
        }
    }


    for (q=j; q<MIN(j+WindowPar+1, NbrExerciseDates);
```

```
q++)
    {
        pnl_mat_get_row(VectToReg, CondExp, q);

        pnl_basis_fit_ls(basis, RegCoeffVect_Iter, Exp
licativeVariables, VectToReg);

        pnl_mat_set_row(RegCoeffMat_Iter, RegCoeffVect_
Iter, q); // Save regression coefficients in RegCoeffMat_Iter.
    }

    for (m=0; m<NbrMCsimulation; m++)
    {
        S_j = MGET(SpotPaths, j, m);
        V_j = MGET(VarPaths, j, m);
        discounted_payoff_j = GET(discount, j) * (p->
Compute)(p->Par, S_j);

        if (discounted_payoff_j>0) // If the discounted
_payoff is null, the OptimalPayoff doesnt change.
        {
            VariablesExplicatives[0] = S_j/S0;
            VariablesExplicatives[1] = V_j/V0;

            regressed_value = 0.;
            for (q=j; q<MIN(j+WindowPar+1, NbrExerciseD
ates); q++)
            {
                pnl_mat_get_row(RegCoeffVect_Iter, Reg
CoeffMat_Iter, q);
                regressed_value = MAX(regressed_value,
pnl_basis_eval(basis, RegCoeffVect_Iter, VariablesExplicati
ves));
            }

            if (discounted_payoff_j >= regressed_value)
            {
                LET(DiscountedOptimalPayoff, m) = dis
counted_payoff_j;
            }
        }
```

```
        }
    }


    /* Price estimator */
    *ptPriceAm_2ndStrategy = MAX((p->Compute)(p->Par, S0),
    pnl_vect_sum(DiscountedOptimalPayoff)/NbrMCsimulation);

    free(VariablesExplicatives);

    pnl_basis_free (&basis);

    pnl_vect_free(&VectToReg);
    pnl_vect_free(&discount);
    pnl_vect_free(&RegCoeffVect_LoSc);
    pnl_vect_free(&RegCoeffVect_Iter);
    pnl_vect_free(&DiscountedOptimalPayoff);

    pnl_mat_free(&ExplicativeVariables);
    pnl_mat_free(&CondExp);
    pnl_mat_free(&SpotPaths);
    pnl_mat_free(&VarPaths);
    pnl_mat_free(&AveragePaths);
    pnl_mat_free(&RegCoeffMat_LoSc);
    pnl_mat_free(&RegCoeffMat_Iter);
    pnl_mat_int_free(&ExerciseDecision);

    return OK;
}

int CALC(MC_AM_Alfonsi_Iterative)(void *Opt, void *Mod,
    PricingMethod *Met)
{
    TYPEOPT* ptOpt=(TYPEOPT*)Opt;
    TYPEMOD* ptMod=(TYPEMOD*)Mod;

    double r,divid;

    r=log(1.+ptMod->R.Val.V_DOUBLE/100.);
    divid=log(1.+ptMod->Divid.Val.V_DOUBLE/100.);
    Met->Par[2].Val.V_INT = MAX(2, Met->Par[2].Val.V_INT);
```

```
// At least two exercise dates.

return MC_Am_Alfonsi_Iterative(
NUMFUNC_1,

LE,

V_DATE-ptMod->T.Val.V_DATE,



PDOUBLE,

al.V_PDOUBLE,

e.Val.V_PDOUBLE,

PDOUBLE,

UBLE,

ENUM.value,

G,

INT,

INT,

ENUM.value,

ENUM.value,

ENUM.value,

INT,

INT,

DOUBLE),
```

```
ptOpt->PayOff.Val.V_

ptMod->S0.Val.V_PDOUB

ptOpt->Maturity.Val.

r,
divid,
ptMod->Sigma0.Val.V_

ptMod->MeanReversion.h

ptMod->LongRunVarianc

ptMod->Sigma.Val.V_

ptMod->Rho.Val.V_PDO

Met->Par[0].Val.V_

Met->Par[1].Val.V_LON

Met->Par[2].Val.V_

Met->Par[3].Val.V_

Met->Par[4].Val.V_

Met->Par[5].Val.V_

Met->Par[6].Val.V_

Met->Par[7].Val.V_

Met->Par[8].Val.V_

&(Met->Res[0].Val.V_
```

```
                                                    &(Met->Res[1].Val.V_
    DOUBLE));
}

static int CHK_OPT(MC_AM_Alfonsi_Iterative)(void *Opt, voi
    d *Mod)
{
    Option* ptOpt=(Option*)Opt;
    TYPEOPT* opt=(TYPEOPT*)(ptOpt->TypeOpt);

    if ((opt->EuOrAm).Val.V_BOOL==AMER)
        return OK;
    else
        return WRONG;
}
#endif //PremiaCurrentVersion

static PremiaEnumMember InitStrategyMembers[] =
{
    { "Andersen-like strategy", 1 },
    { "Longstaff-Schwartz strategy", 2 },
    { NULL, NULLINT }
};

static DEFINE_ENUM(InitStrategy,InitStrategyMembers);

static int MET(Init)(PricingMethod *Met,Option *Opt)
{
    if ( Met->init == 0)
    {
        Met->init=1;

        Met->Par[0].Val.V_ENUM.value=1;
        Met->Par[0].Val.V_ENUM.members=&InitStrategy;
        Met->Par[1].Val.V_LONG=50000;
        Met->Par[2].Val.V_INT=10;
        Met->Par[3].Val.V_INT=1;
        Met->Par[4].Val.V_ENUM.value=0;
        Met->Par[4].Val.V_ENUM.members=&PremiaEnumRNGs;
        Met->Par[5].Val.V_ENUM.value=2;
        Met->Par[5].Val.V_ENUM.members=&PremiaEnumCirOrder;
```

```
        Met->Par[6].Val.V_ENUM.value=0;
        Met->Par[6].Val.V_ENUM.members=&PremiaEnumBasis;
        Met->Par[7].Val.V_INT=10;
        Met->Par[8].Val.V_INT=5;
    }

    return OK;
}


PricingMethod MET(MC_AM_Alfonsi_Iterative)=
{
    "MC_AM_Alfonsi_Iterative",
    {
        {"Initial Strategy",ENUM,{100},ALLOW},
        {"N Simulations",LONG,{100},ALLOW},
        {"N Exercise Dates",INT,{100},ALLOW},
        {"N Steps per Period",INT,{100},ALLOW},
        {"RandomGenerator",ENUM,{100},ALLOW},
        {"Cir Order",ENUM,{100},ALLOW},
        {"Basis",ENUM,{100},ALLOW},
        {"Dimension Approximation",INT,{100},ALLOW},
        {"Window Parameter",INT,{100},ALLOW},
        {" ",PREMIA_NULLTYPE,{0},FORBID}},
    CALC(MC_AM_Alfonsi_Iterative),
    {   {"Price by 2nd Strategy",DOUBLE,{100},FORBID},
        {"Price by 1st  Strategy",DOUBLE,{100},FORBID},
        {" ",PREMIA_NULLTYPE,{0},FORBID}},
    CHK_OPT(MC_AM_Alfonsi_Iterative),
    CHK_ok,
    MET(Init)
};
```

References