```
/****************************************************************
 * Source file             : c_GConvert.cpp
 * File summary            : Conversion processing main file
 * Vertion                 : 1.4
 * Created by              :
 * Updated on (created on) : 2007.04.05(2002.10.01)
 * Remarks                 : Compile switches for compiling are listed below.
 *                         : GUS
 *    _MSC_VER             : Microsoft Visual C++ ver 6 or over
 *    _GNUC_               : GNU C++/GCC/G++
 *                         : Borland C++Builder 5 or over
 * HISTORY                 :
 * ID  -- DATE -- ----- NOTE -----------------------------------
 * 00  2002.10.01 First release
 *   ----- V1.2 ----
 * 01  2005.08.30 Calculation with 5 shift if 6 shift is not equipped
 * 02  2005.08.30 MaxCarA calculation in starting process is recovered
 *   ----- V1.3 ----
 * 03  2005.02.28 Idle rpm with not re-calc if not started
 *   ----- V1.4 ----
 * 04  2007.04.05 Added 3second maintenance algorithm
 * 05  2007.07.02 first time is max trq. and shift logic recoverd
 ***************************************************************/
#ifndef __CONVERT__
#define __CONVERT__
#define MY_VERSION "1.4"

//============================================================================
// Include
//============================================================================
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <iostream>
#include <map>
#include <set>
#include <vector>
#include <string>
using namespace std;

//--------------------------------
// Environmental switch
//--------------------------------
#ifdef __GNUC__
#define __STL_HAS_NAMESPACES
#endif

#pragma hdrstop

//--------------------------------
// Environmental switch
//--------------------------------
#ifndef _MSC_VER
#pragma package(smart_init)
#else
#pragma optimize("g",off)
#endif

//============================================================================
// Constant declaration (define)
//============================================================================
//--------------------------------
// Output data headers
//--------------------------------
#define DEF_PRINT_POS1           "time(s)"
#define DEF_PRINT_POS2           "Vtarget(km/h)"
#define DEF_PRINT_POS3           "Vreal(km/h)"
#define DEF_PRINT_POS4           "Ne(rpm)"
#define DEF_PRINT_POS5           "Te(N-m)"
#define DEF_PRINT_POS6           "N_norm(%)"
#define DEF_PRINT_POS7           "T_norm(%)"
#define DEF_PRINT_POS8           "Shift"

//--------------------------------
// Software configuration values
```

```
//----------------------------------
#define DEF_MAIN_ENVFILE         "DATA"  // Default name of the file containing environment values

#define LINE_MAX_LENGTH          1024    // The number max of caracter per line
#define DATA_FILES_NUMBER        3       // Data are stored in three files (env,spec,torque)

#define NG                       -1      // Basic negative value meaning some error occured
#define OK                       1       // Basic positive value meaning a success

//----------------------------------
// System physical values
//----------------------------------
#define GEAR_HOLD_TIME           3       // Gear hold time in internal data (sec)

#define UD                       0.95    // Select optimum gear: Sets final reduction ratio (transmission
efficiency) to fixed value 0.95.
#define E_FACT                   0.03    // Inertial weight ratio equivalent in rotation section (E_FACT)
#define M_FACT                   0.07    // Inertial weight ratio equivalent in rotation section (M_FACT)
#define PERSON_W                 55      // Weight per person  (55kg)

#define CLUTCH_MEET              5       // Sets clutch meet normalized revolution in internal data
#define PI                       3.14    // Circle circumference ratio to diameter
#define G                        9.8     // Gravitational acceleration

#define DEF_FORCE_ON98           0.98    // 98%: Gear transmission efficiency
#define DEF_FORCE_OFF95          0.95    // 95%: Gear transmission efficiency


//----------------------------------
// Specification default values
//----------------------------------
#define DEF_MAXGEAR              7       // Default number of gear positions
#define DEF_GEAR_RATIO           1
#define DEF_FINAL_REDUC_RATIO    4.711
#define DEF_IDLING_ENGINE_SPEED  500
#define DEF_MAX_OUTPUT_RATIO     3100


//----------------------------------
// Output messages for error codes
//----------------------------------
#define MSG_WRITE_FILE_ERROR             "File write error."

#define ERROR_MAIN_FILE_NOT_FOUND_STR    "Main configuration file has not been found."
#define ERROR_ENV_FILE_NOT_FOUND_STR     "Environment file could not be found."
#define ERROR_SPEC_FILE_NOT_FOUND_STR    "Specification file could not be found."
#define ERROR_TORQUE_FILE_NOT_FOUND_STR  "Torque data file could not be found."
#define ERROR_ENV_FILE_EMPTY_STR         "Environment file seems to be empty."
#define ERROR_SPEC_FILE_EMPTY_STR        "Specification file seems to be empty."
#define ERROR_TORQUE_FILE_EMPTY_STR      "Torque data file seems to be empty."
#define ERROR_SPEC_DATA_FORMAT_STR       "Specification file seems to have a wrong format."

//----------------------------------
// Error codes
//----------------------------------
#define ERROR_MAIN_FILE_NOT_FOUND        0
#define ERROR_ENV_FILE_NOT_FOUND         -1
#define ERROR_SPEC_FILE_NOT_FOUND        -2
#define ERROR_TORQUE_FILE_NOT_FOUND      -3
#define ERROR_ENV_FILE_EMPTY             -5
#define ERROR_SPEC_FILE_EMPTY            -6
#define ERROR_TORQUE_FILE_EMPTY          -7
#define ERROR_SPEC_DATA_FORMAT           -10


//----------------------------------
// Different flag values used to qualify the engine behaviour
//----------------------------------
#define ENGINE_IDLE                      0       // IDLE
#define ENGINE_START                     1       // Start
#define ENGINE_CONSTANT                  2       // Constant speed
#define ENGINE_DECELERATE                3       // Decelerate (including shift-down)
#define ENGINE_ACCELERATE                4       // Accelerate (including shift-up)
```

```
//==============================================================================
// Structure declaration
//==============================================================================

//--------------------------------
// Structure for Torque Input data save
//--------------------------------
struct stTorqueData
{
    double  d_EngineRevolutions;         // Engine speed  (rpm)
    double  d_EngineTorque;              // Engine torque (Nm)
};

//--------------------------------
// Structure for analysis processing
//--------------------------------
typedef struct stCalculateData{
    double  fTimes;                      // Accumulated time (msec)[for analysis]
                                         // (sec) [for read]
    int     nCalcGear;                    // Calculated Gear
    int     nGearTime;                   // Gear determination time duration (msec)
    int     nCalcTime;                   // Required time
    int     nFlag;                       // Holds same flag due to use of previous version
                                         //  ENGINE_IDLE:           IDLE
                                         //  ENGINE_START:          Start
                                         //  ENGINE_CONSTANT_SPEED: Constant speed
                                         //  ENGINE_DECELERATE:     Decelerate (including shift-down)
                                         //  ENGINE_ACCELERATE:     Accelerate (including shift-up)
    double  fVTarget_sp;                 // Target speed (km/h)
    double  fVAna_sp;                    // Calculated vehicle speed
    double  fNeRevo;                     // Engine speed
    double  fTe;                         // Engine torque
    bool    bClutchMeetMode;             // If clutchMeet mode is engaged or not
}stCalculateData;

//--------------------------------
// Structure for optimal gear search
//--------------------------------
typedef struct stOptimalGears{
    int iGearsNb;
    int iGearsID[DEF_MAXGEAR];
    double dMaintainTime[DEF_MAXGEAR];
    double dDifferenceSpped[DEF_MAXGEAR];
    bool bTargetSpeedFollowed[DEF_MAXGEAR];
    bool bGearPatternFollowed[DEF_MAXGEAR];
    bool bBestMaintainTime[DEF_MAXGEAR];
    bool bGearChangeNeeded[DEF_MAXGEAR];
}stOptimalGears;

//==============================================================================
// Class declaration
//==============================================================================
class TCalculateProc
{
public:
    TCalculateProc();                           // Constructor
    virtual ~TCalculateProc();                  // Destructor

    //Function declaration
    bool Init();                                // Analysis processing initialization
    bool Init(string OutputFileName);           // Analysis processing initialization (with a known output
file name)
    bool DataClear();                           // Analysis data clear processing

    //--------------------------------
    // Input/Output filenames setting
    //--------------------------------
    void setInputFileName(string s){m_sInputFileName=s;}
    void setOutputFileName(string s){m_sOutputFileName=s;}

    string getInputFileName(){return m_sInputFileName;}
    string getOutputFileName(){return m_sOutputFileName;}

    //--------------------------------
    // Functions retreiving data from files
    //---------------------------------
    int Data_Acquisition();                    // Checks main file's presence then launch data copy
    int SaveEnvironmentData(string fileName);   // Stores Environment data
    int SaveSpecificationData(string fileName); // Stores Specification data
    int SaveTorqueData(string fileName);        // Stores Torque data


    //---------------------------------
    // Calcultaion main steps' functions
    //---------------------------------
    int  CalculateProcess();                   // Initiates analysis processing.

    bool Calculate_GearUpMode();               // Detects if GearUp mode is required
    bool Calculate_EngineBehaviourFlag();      // Determines flag for pattern compatible with previous
version.
    bool Calculate_progress();                 // Sets gear (according to revolution).

    //---------------------------------
    // Output functions used to store values after calculation process
    //---------------------------------
    void   DispCalculateData();                // Displaying calculated datas
    int    WriteAllCalculateData();            // Storing those data in output file


    //---------------------------------
    // Calculation detailed steps' functions
    //---------------------------------
    bool Calculate_Engine_IDLE(
                map<double,stCalculateData>::iterator p_first,
                map<double,stCalculateData>::iterator p_second );   // IDLE section processing

    bool Calculate_Engine_START(
                map<double,stCalculateData>::iterator &p_first);    // Processing until analysis vehicle
speed reaches reference vehicle speed

    bool Calculate_Engine_ACCELERATE(
                map<double,stCalculateData>::iterator p_first,
                map<double,stCalculateData>::iterator &p_second );  // Section setting for acceleration

    bool Calculate_Engine_DECELERATE(
                map<double,stCalculateData>::iterator p_first,      // Section setting for deceleration
                map<double,stCalculateData>::iterator p_second );

    //---------------------------------
    // Vector storing analysis data
    //---------------------------------
    map<double,stCalculateData> setCalculateData;                   // Analysis data table
    map<double,stCalculateData>::iterator p_setCalculateData;       // Analysis data pointer

private:

        //---------------------------------
    // Best gear determination function
    //---------------------------------
        stOptimalGears GetBestEngineSpeedMaintainGear(map<double,stCalculateData>::iterator p_start,
                                        map<double,stCalculateData>::iterator p_end,
                                        int iAskedGear,
                                        double dGearHoldTime,
                    double dHoldTimes=0);

    //---------------------------------
    // Function calculating gear maintain time
    //---------------------------------
        double GetGearMaintainTime(     map<double,stCalculateData>::iterator p_start,
                                                            map<double,stCalculateData>::iterator
p_end,
                                                            int iGear,
                                                            double dGearHoldTime,
                                bool &bTargetSpeedFollowed,
                                bool &bGearPatternFollowed,
                                bool &bGearChangeNeed,
                                double &dDifferenceSpeed,
                                int  iShiftChangeTimes =0);

    //-------------------------------------
    // Saving results when gear is maintained
    //---------------------------------------
    map<double,stCalculateData>::iterator RecordFixedGear(
                            map<double,stCalculateData>::iterator p_start,
                            map<double,stCalculateData>::iterator p_end,
                            int iGear,
                            double dMaintainTime);

    //--------------------------------
    // Process used during initialization
    //--------------------------------
    double GetGearPass(int nGear);             // Obtains gear transmission efficiency.
    double GetLineReviseMaxTorque(double fNe); // Obtains max. torque data, and executes calculation.

    bool   CalcTeMaxSp(int nGear, double fTm,
                   double fPrevV, double &fV,
                   double &fNe, double &fTe);  // Calculates the higher speed when torque exceeds maximum
limit

    //--------------------------------
    // Calculation logic
    //--------------------------------
    double calcRL (double fV);                         // Calculates rolling resistance.
    double GetCarWeight(bool bFlag, int nGear=0);      // Reads and calculates vehicle body weight.

    int GetNe(int nGear, double fVg, double &fNe);     // Obtains revolution.
    int GetV(int nGear, double fNe, double &fVg);      // Obtains Speed.
    int GetTe(int nGear,double fTargetSp,
              double fCarAcc,double &fTe);             // Calculates torque

    //--------------------------------
    // Saving calculated datas
    //--------------------------------
    int WriteHead(FILE *fp);                   // HEAD file write processing


    //=========================================================================
    // Environment parameters
    //=========================================================================
    string m_sInputFileName;                   // Input file name
    string m_sOutputFileName;                  // Output file name


    //--------------------------------
    // Vehicle specification settings
    //--------------------------------
    double  m_fCarIniW;                        // Empty vehicle mass (kg)
    double  m_fCarPayload;                     // Payload of car (kN)
    double  m_fPersons;                        // Riding capacity (in number of persons)
    double  m_fOverHeight;                     // Overall vehicle height
    double  m_fOverWidth;                      // Overall vehicle width
    double  m_fTireRollRadius;                 // Tire rolling radius
    int     m_nMaxGear;                        // Max. number of gears
    vector<double> m_vGearRatio;               // Ratio of each gear
    double  m_fLastReduceGear;                 // Final reduction ratio
    double  m_fIdleSpeed;                      // Idling (IDLE) revolution
    double  m_fMaxOutputRotation;              // Max. torque revolution[rpm]

    //--------------------------------
    // JARI mode
    //--------------------------------
    double  m_fClutch_MeetNe;                  // Clutch meet revolution

    //--------------------------------
    //Other member variables
    //--------------------------------
    int     m_nPtnGearUp;                      // Gear ref-mode 0:normal 1:gear pos.+1

    //=========================================================================
    // Others calculation parameters
    //=========================================================================
    //--------------------------------
    // Vectors storing input datas:
    //--------------------------------
    vector<double> m_vSpecificationData;       // Vector containing specification datas
    //--------------------------------
    // Torque linked elements
    //--------------------------------
    set<stTorqueData> m_MaxTorque;              // Max. torque data
    set<stTorqueData>::iterator p_MaxTorque;    // Pointer


    friend bool operator<(const stTorqueData& a,const stTorqueData& b){// Max. torque data operator
        // Uniquely sorted by engine speed.
        return( a.d_EngineRevolutions < b.d_EngineRevolutions );
    };
};


//============================================================================
// Class declaration
//============================================================================
TCalculateProc *CalculateProc;

/**/
/****************************************************************
 * Function name          : TCalculateProc
 * Function summary       : Constructor
 * Explanation            : Class constructor
 *                        :
 * Argument (input)       : None
 * Argument (output)      : None
 * Argument (I/O)         : None
 * Return value           : None
 * Created by             :
 * Updated on (created on) :
 * Remarks                :
 ****************************************************************/
TCalculateProc::TCalculateProc()
{
    m_sOutputFileName = "";           // Initializes output file name
    return;
}
/**/
/****************************************************************
 * Function name          : ~TCalculateProc
 * Function summary       : Destructor: Clears all analysis data
 * Explanation            : Class destructor
 *                        :
 * Argument (input)       : None
 * Argument (output)      : None
 * Argument (I/O)         : None
 * Return value           : None
 * Created by             :
 * Updated on (created on) :
 * Remarks                :
 ****************************************************************/
TCalculateProc::~TCalculateProc()
{
    // -----------------------
    //Deletion of setCalculateData vector
    // -----------------------
    if(setCalculateData.empty() != true){
        setCalculateData.erase(setCalculateData.begin(), setCalculateData.end());
        setCalculateData.clear();
    }

    // -----------------------
    //Deletion of m_vGearRatio vector
    // -----------------------
    if(m_vGearRatio.empty() != true){
        m_vGearRatio.erase(m_vGearRatio.begin(), m_vGearRatio.end());
        m_vGearRatio.clear();
    }

    // -----------------------
    //Deletion of m_vSpecificationData vector
    // -----------------------
    if(m_vSpecificationData.empty() != true){
        m_vSpecificationData.erase(m_vSpecificationData.begin(), m_vSpecificationData.end());
        m_vSpecificationData.clear();
    }

    return;
}
/**/
/****************************************************************
 * Function name          : Init
 * Function summary       : Analysis processing initialization (with output file provided)
 * Explanation            : Convert processing is initialized (with output file provided).
 *                :
 * Argument (input)       : FineName: Output file name
 * Argument (output)      : None
 * Argument (I/O)         : None
 * Return value           : true : Normal    false : Failure
 * Created by             :
 * Updated on (created on) :
 * Remarks                :
 ****************************************************************/
bool TCalculateProc::Init(string OutputFileName)
{
    bool    bRet;

    m_sOutputFileName = OutputFileName;
    bRet = Init();

    return(bRet);                     // Returns if Init() was successful
}
/**/
/****************************************************************
 * Function name          : Init
 * Function summary       : Analysis processing initialization
 * Explanation            : Convert processing initialization
 *
 * Argument (input)       : None
 * Argument (output)      : None
 * Argument (I/O)         : None
 * Return value           : true : Normal    false : Failure
 * Created by             :
 * Updated on (created on) :
 * Remarks                :
 ****************************************************************/
bool TCalculateProc::Init()
{
    // ----------------------
    // Setting of vehicle body weight and riding capacity weight data
    // ----------------------
    m_fCarIniW =        (double)m_vSpecificationData[0] ;   // Sets empty vehicle mass (kg).
    m_fCarPayload =     (double)m_vSpecificationData[1]/2; // Obtain test payload of car (kg) (max.
payload/2)
    m_fPersons =        (double)m_vSpecificationData[2];   // Riding capacity (in number of persons)
    m_fOverHeight =     (double)m_vSpecificationData[3];   // Overall vehicle height
    m_fOverWidth =      (double)m_vSpecificationData[4];   // Overall vehicle width
    m_fTireRollRadius=  (double)m_vSpecificationData[5];   // Tire dynamic rolling radius
    m_nMaxGear =        (int)m_vSpecificationData[6];      // Top gear (Number of gear position)

    // ----------------------
    // Reads gear ratio
    // ----------------------
    for( int i = 1; i <= m_nMaxGear; i++ ){
        m_vGearRatio.push_back((double)m_vSpecificationData[6+i]);   // Stores gear ratio.
    }

    m_fLastReduceGear = (double)m_vSpecificationData[7+m_nMaxGear];  // Final reduction ratio
    m_fIdleSpeed =      (double)m_vSpecificationData[8+m_nMaxGear];  // Idling engine speed
    m_fMaxOutputRotation=(double)m_vSpecificationData[9+m_nMaxGear]; // Max. output revolution


    // ----------------------
    //Calculates clutch meet engine speed
    // ----------------------
    m_fClutch_MeetNe=(m_fMaxOutputRotation-m_fIdleSpeed)*CLUTCH_MEET/100.0+m_fIdleSpeed;

    return(true);
```

```
}
/**/
/****************************************************************
 * Function name           : CalculateProcess
 * Function summary        : Main processing of Convert
 * Explanation             : Main processing of Convert is executed.
 *                         :
 * Argument (input)        : None
 * Argument (output)       : None
 * Argument (I/O)          : None
 * Return value            : OK : Normal   NG: Failure
 * Created by              :
 * Updated on (created on) :
 * Remarks                 :
 ****************************************************************/
int TCalculateProc::CalculateProcess()
{
    // Detects engine's behaviour
    if(!Calculate_EngineBehaviourFlag()) return NG;

    // Detects if Gear up is necessary
    if(!Calculate_GearUpMode()) return NG;

    // Determines gear, and sets parameters
    if(!Calculate_progress()) return NG;

    // Save calculated datas
    DispCalculateData();         // Displays parameter information
    WriteAllCalculateData();     // Data write

    return OK;
}

/**/
/****************************************************************
 * Function name           : Calculate_GearUpMode
 * Function summary        : Look for a gearUp mode need and apply it if required
 * Explanation             : In input datas, if one speed exceeds the engine limit, ...
 *                         : ... all gears (except null and maximum gear) are incremented.
 * Argument (input)        : None
 * Argument (output)       : None
 * Argument (I/O)          : None
 * Return value            : true : Normal    false : Failure
 * Created by              :
 * Updated on (created on) :
 * Remarks                 :
 ****************************************************************/
bool TCalculateProc::Calculate_GearUpMode()
{
    bool   bPtnGearUp_Flg = false;                                  // Temporary gear-up mode flag
    double tmpNe;                                                   // Temporary revolution
    int    nRet;
int tmpGear;

    //--------------------------------
    // Looking for m_fMaxOutputRotation exceding case
    //--------------------------------
    for(p_setCalculateData = setCalculateData.begin(); p_setCalculateData!=setCalculateData.end();
p_setCalculateData++)
    {
        if( p_setCalculateData->second.nCalcGear != 0 ){
            nRet = GetNe( p_setCalculateData->second.nCalcGear, p_setCalculateData->second.fVTarget_sp,
tmpNe); // Engine speed
            if( nRet != OK ){
                return( false );
            }
            if( tmpNe > m_fMaxOutputRotation ){                    // Test if Max. output revolution is
exceeded
                bPtnGearUp_Flg = true;                             // Pattern gear-up mode setting
                break;
            }
        }
    }

    //--------------------------------
    // In case of m_fMaxOutputRotation exceding case, pattern gear-up mode is active: all gears are
upgraded
    //--------------------------------
    if(bPtnGearUp_Flg){
        m_nPtnGearUp = 1;


for(p_setCalculateData=setCalculateData.begin();p_setCalculateData!=setCalculateData.end();p_setCalculateDa
ta++){                                 // Loops for number of analysis data items.
            if(p_setCalculateData->second.nCalcGear != 0 ){              // If gear is described

                if(p_setCalculateData->second.nCalcGear+1<=m_nMaxGear){ // If gear-up value is allowed
                    p_setCalculateData->second.nCalcGear++;              // Executes gear-up.
                }
            }
        }
    }
    else{
        m_nPtnGearUp = 0;
    }
    return( true );
}

/**/
/*****************************************************************
 * Function name           : Calculate_EngineBehaviourFlag
 * Function summary        : Refering to the evolutions of speed engine's behaviour, ...
 * Explanation             : ... engine's mode is calculated and flag values are set
 *                         :
 * Argument (input)        : None
 * Argument (output)       : None
 * Argument (I/O)          : None
 * Return value            : true : Normal    false : Failure
 * Created by              :
 * Updated on (created on) :
 * Remarks                 :
 *****************************************************************/
bool TCalculateProc::Calculate_EngineBehaviourFlag()
{
    int n_prevFlag;                                                 // Previous flag
    double fprev_V;                                                 // Previous speed

    n_prevFlag = ENGINE_IDLE;                                       // Initializes previous flag
    fprev_V = 0;                                                    // Initializes previous speed

    //--------------------------------
    // Loops over all analysis data items:
    //--------------------------------
    for(p_setCalculateData = setCalculateData.begin(); p_setCalculateData != setCalculateData.end();
p_setCalculateData++){
        //--------------------------------
        // If same as previous speed
        //--------------------------------
        if(p_setCalculateData->second.fVTarget_sp == fprev_V){
            if(n_prevFlag == ENGINE_IDLE){                          // If previous operation is IDLE state.
                p_setCalculateData->second.nFlag = ENGINE_IDLE;     //  ->IDLE state is kept.
            }
            else{                                                   // If other than IDLE state
                p_setCalculateData->second.nFlag = ENGINE_CONSTANT; //  ->Speed is constant
            }
        }
        //--------------------------------
        // If faster than previous speed
        //--------------------------------
        else if(p_setCalculateData->second.fVTarget_sp > fprev_V){
            if(p_setCalculateData == setCalculateData.begin()){     // In case of first data not null
                p_setCalculateData->second.nFlag=ENGINE_CONSTANT;   //  ->Executes constant speed
processing
            }
            else if(n_prevFlag == ENGINE_IDLE){
                p_setCalculateData->second.nFlag = ENGINE_START;    // Sets flag in starting data.
            }
            else
                p_setCalculateData->second.nFlag=ENGINE_ACCELERATE; // Acceleration state
        }
        //-------------------------------
        // If slower than previous speed
        //-------------------------------
        else{
            if(p_setCalculateData->second.fVTarget_sp == 0){        // If speed is 0
                p_setCalculateData->second.nFlag = ENGINE_IDLE;     //  ->Sets to IDLE state for this time.
            }
            else{                                                   // If speed is not 0
                p_setCalculateData->second.nFlag=ENGINE_DECELERATE; //  -> Operation for this time is
deleceration
            }
        }

        n_prevFlag = p_setCalculateData->second.nFlag;              // Holds flag for this time
        fprev_V    = p_setCalculateData->second.fVTarget_sp;        // Holds speed for this time
    }
    return( true );
}
/**/
/*****************************************************************
 * Function name          : Calculate_progress
 * Function summary       : Processing data setup processing
 * Explanation            : According to each processing method, applicable module is initiated
 *                        : and data is set
 * Argument (input)       : None
 * Argument (output)      : None
 * Argument (I/O)         : None
 * Return value           : true : Normal    false : Failure
 * Created by             :
 * Updated on (created on) :
 * Remarks                :
 *****************************************************************/
bool TCalculateProc::Calculate_progress()
{
    map<double,stCalculateData>::iterator p_tmp;
    map<double,stCalculateData>::iterator p_first;                          // First data
    map<double,stCalculateData>::iterator p_second;                         // Next data
    bool bRet;                                                              // Function return value
    int tmpSize, tmpNow;                                                    // Percentage
    char buf[256];

    tmpSize = (int)setCalculateData.size();                                 // Sets size.
    tmpNow = 0;

    //-------------------------------
    // Make gear settings for all analysis data items
    //-------------------------------
    for( p_setCalculateData = setCalculateData.begin();
            p_setCalculateData != setCalculateData.end();
            p_setCalculateData++ ){                                         // Checks all analysis data
items.

        //-------------------------------
        // Determine target range.
        //-------------------------------
        p_first = p_setCalculateData;                                       // Sets first range position.
        for( p_second = p_first; p_second->second.nFlag == p_first->second.nFlag && p_second !=
setCalculateData.end();p_second++ )
        {                               // Up to same flag
            tmpNow++;                                                       // Increments count.
        }

        sprintf( buf, "¥b¥b¥b¥b¥b¥b%5.1f%%", (double)tmpNow / (double)tmpSize * 100.0 );
        cout << buf;
        if( tmpSize == tmpNow ){
            cout << "¥b¥b¥b¥b¥b      ";
            cout << endl;
        }


        switch( p_setCalculateData->second.nFlag ){                         // Sets gear according to
pattern information flag.

            //-------------------------------
            // Starting gear setting
            //--------------------------------
            case ENGINE_START:
                bRet = Calculate_Engine_START(p_setCalculateData);          // Post-processing for vehicle
start
                if( bRet == false ){
                    return false;
                }
                break;

            //--------------------------------
            // In case of IDLE state
            //--------------------------------
            case ENGINE_IDLE:
                bRet = Calculate_Engine_IDLE(p_first, p_second);
                if( bRet == false ){
                    return false;
                }
                p_setCalculateData = p_second;
                p_setCalculateData--;
                break;

            //--------------------------------
            // Gear setting for constant speed running
            //--------------------------------
            case ENGINE_CONSTANT:
                bRet = Calculate_Engine_ACCELERATE( p_first, p_second );  // Sets gear for constant speed
running.
                if( bRet == false ){
                    return false;
                }
                p_setCalculateData = p_second;
                p_setCalculateData--;
                break;


            //--------------------------------
            // Gear setting for deceleration
            //--------------------------------
            case ENGINE_DECELERATE:
                bRet = Calculate_Engine_DECELERATE( p_first, p_second );  // Sets gear for constant speed
running.
                if( bRet == false ){
                    return false;
                }
                p_setCalculateData = p_second;
                p_setCalculateData--;
                break;


            //--------------------------------
            // Gear setting for acceleration
            //--------------------------------
            case ENGINE_ACCELERATE:
                bRet = Calculate_Engine_ACCELERATE(p_first, p_second ); // Sets free-running time for
acceleration.
                if( bRet == false ){
                    return false;
                }
                p_setCalculateData = p_second;
                p_setCalculateData--;
                break;
        }
    }

    cout << "¥b¥b¥b¥b¥b¥b             " << endl;

    return true;
}

/**/
/****************************************************************
 * Function name          : Calculate_Engine_IDLE
 * Function summary       : Idle section setup processing
 * Explanation            : Settings are made for idling section.
 *                         :
 * Argument (input)        : p_first   : First pointer
 * Argument (input)        : p_second  : Next setting pointer
 * Argument (output)       : None
 * Argument (I/O)          : None
 * Return value            : true : Normal    false : Failure
 * Created by              :
 * Updated on (created on) :
 * Remarks                 :
 ****************************************************************/
bool TCalculateProc::Calculate_Engine_IDLE(map<double,stCalculateData>::iterator p_first,
                                           map<double,stCalculateData>::iterator p_second )
{
    while( p_first != p_second ){               // Setting IDLE properties for each data between first and
second                  // Loops according to range.
        p_first->second.fNeRevo = m_fIdleSpeed; // Sets revolution.
        p_first->second.fTe = 0;                // Sets engine torque.
        p_first++;                              // Next position processing
    }
    return true;
}

/**/
/****************************************************************
 * Function name           : Calculate_Engine_START
 * Function summary        : Starting target-speed follow processing
 * Explanation             : Settings are made for the case in which target-speed follow is impossible
 *                         : during vehicle start.
 * Argument (input)        : None
 * Argument (output)       : None
 * Argument (I/O)          : p_first   : First pointer
 * Return value            : true : Normal    false : Failure
 * Created by              :
 * Updated on (created on) :
 * Remarks                 :
 ****************************************************************/
bool TCalculateProc::Calculate_Engine_START(map<double,stCalculateData>::iterator &p_first )
{
    int     nRet;                                       // Return value of function to be recalled
    int     i;
    map<double,stCalculateData>::iterator p_prevData;   // Temporary pointer

    double tmpVana_sp;                                  // Calculated speed
    double tmpTeMax;                                    // Maximum torque

    int prevGearTime;                                   // Gear required time
    int prevCalcTime;                                   // Required time
    double prevAcc;                                     // Previous acceleration

    double fMaxAcc;                                      // Max. acceleration
    double fCarAcc;                                      // Set acceleration
    double fNeRevo;                                      // Engine speed
    double fTe;                                          // Engine torque
    int tmpNowGear;
    bool bTmp;
    bool bTmp2;
    bool bGearChangeNeed;
    double ftmp4;
    double dCurrentGearMaintainTime;

    p_prevData = p_first;
    p_prevData--;

    prevGearTime = p_prevData->second.nGearTime;         // Previous gear time
    prevCalcTime = p_prevData->second.nCalcTime;         // Previous required time
    tmpVana_sp=p_first->second.fVTarget_sp;

    // Calculates current gear maintenance time
    tmpNowGear = p_first->second.nCalcGear;
    for( i =  GEAR_HOLD_TIME; i>= 1; i-- ){
        dCurrentGearMaintainTime = GetGearMaintainTime( p_first, setCalculateData.end(), tmpNowGear, i,
bTmp, bTmp2, bGearChangeNeed, ftmp4);
        if( bGearChangeNeed == false ){
            break;
        }
    }

    if(dCurrentGearMaintainTime>0)
    {
        p_first = RecordFixedGear(p_first, setCalculateData.end(), tmpNowGear, dCurrentGearMaintainTime);
        return true;
    }

    //--------------------------------
    // Calculates engine speed (rpm)
    //--------------------------------
    GetNe(p_first->second.nCalcGear, p_first->second.fVTarget_sp ,fNeRevo);

    if(fNeRevo<m_fClutch_MeetNe && p_prevData->second.fVAna_sp==0){
        fNeRevo=m_fClutch_MeetNe;
        p_first->second.bClutchMeetMode=true;
    }

    //--------------------------------
    // Calculates engine torque
    //--------------------------------
    fCarAcc = (p_first->second.fVTarget_sp - p_prevData->second.fVAna_sp)/(prevCalcTime/10) /3.6;
    GetTe(p_first->second.nCalcGear,p_first->second.fVTarget_sp,fCarAcc, fTe);
    tmpTeMax = GetLineReviseMaxTorque(fNeRevo);

    if(fTe>tmpTeMax){//Here the target speed value will be changed
        CalcTeMaxSp(p_first->second.nCalcGear, prevCalcTime, p_prevData->second.fVAna_sp, tmpVana_sp,
fNeRevo, fTe );
    }

    //--------------------------------
    // Saving starting results:
    //--------------------------------
    p_first->second.fVAna_sp  = tmpVana_sp;              // Sets analysis speed.
    p_first->second.fNeRevo   = fNeRevo;                 // Sets engine speed.
    p_first->second.fTe       = fTe;                     // Sets engine torque.
    p_first->second.nGearTime = prevGearTime + prevCalcTime;

    return true;
}

/**/
/*****************************************************************
 * Function name          : Calculate_Engine_ACCELERATE
 * Function summary       : Calculates gear, speed and torque when car accelerates
 * Explanation            :
 *                        :
 * Argument (input)       : p_first   : First pointer of acceleration period
 * Argument (input)       : p_second  : last pointer of acceleration period
 * Argument (output)      : None
 * Argument (I/O)         : None
 * Return value           : true : Normal    false : Failure
 * Created by             :
 * Updated on (created on) :
 * Remarks                :
 *****************************************************************/
bool TCalculateProc::Calculate_Engine_ACCELERATE(map<double,stCalculateData>::iterator p_first,
                        map<double,stCalculateData>::iterator &p_second )
{
    map<double,stCalculateData>::iterator p_tmpCalculate;      // Temporary pointer
    map<double,stCalculateData>::iterator p_previous;          // Previous data

    int tmpNowGear;                                            // Temporary gear
    int tmpNowChangePrevGear;                                  // Temporary gear
    int tmpPrevGear;                                           // Previous gear
    int iContinueSettingGear;

    double tmpNe;                                              // Calculated engine speed

    double fCarAcc;                                            // Acceleration value (used to calculate
torque)
    double tmpTe;                                              // Calculated torque
    double tmpTeMax;                                           // Maximum torque

    double tmpTargetSpeed;                                     // Calculated speed
    double tmpPrevVAna_sp;                                     // Previous analysis speed
    double tmpPrevGearTime;                                    // Gear required time
    double tmpPrevCalcTime;                                    // Required time

    bool bCurrentGearSpeedFollowed;
    bool bCurrentGearPatternFollowed;
    bool bCurrentGearChangeNeeded;
    double dDifferenceSpeed;
    double dRemainingTime;
    double dCurrentGearMaintainTime;
    bool bNowChangePrevGearSpeedFollowed;
    bool bNowChangePrevGearPatternFollowed;
    bool bNowChangePrevGearChangeNeeded;
    double dNowChangePrevDifferenceSpeed;
    double dNowChangePrevGearMaintainTime;
    stOptimalGears optimalGears;

    //--------------------------------
    // Initialization of first gear value
    //--------------------------------
    p_previous = p_first;
    p_previous--;
    tmpNowGear = p_previous->second.nCalcGear;
    if(tmpNowGear==0) tmpNowGear = p_first->second.nCalcGear;

    for(p_tmpCalculate=p_first ; p_tmpCalculate!=p_second ; p_tmpCalculate++){

        tmpTargetSpeed  = p_tmpCalculate->second.fVTarget_sp;

        p_previous = p_tmpCalculate;
        p_previous--;

        tmpPrevGearTime = p_previous->second.nGearTime;        // Previous gear time
        tmpPrevCalcTime = p_previous->second.nCalcTime;        // Previous required time
        tmpPrevVAna_sp  = p_previous->second.fVAna_sp;         // Previous analysis speed
        tmpPrevGear     = p_previous->second.nCalcGear;

        if(p_tmpCalculate!=p_first){
            tmpNowGear = p_previous->second.nCalcGear;
        }

        memset( &optimalGears, 0x00, sizeof( optimalGears ));

        bool bGearChange = false;

        //--------------------------------
        // Test if gear change has not been done too recently (shortest period, in sec, is GEAR_HOLD_TIME)
        //--------------------------------
        if( ((int)((tmpPrevGearTime + tmpPrevCalcTime)/10.0)) >GEAR_HOLD_TIME){
            //--------------------------------
            // In case of deceleration previously
            //--------------------------------
            if(p_previous->second.nFlag == ENGINE_DECELERATE){
                if((tmpTargetSpeed<10.0)&&(tmpNowGear>1+m_nPtnGearUp)&&(1+m_nPtnGearUp<=m_nMaxGear)){
                    tmpNowGear = min(1 + m_nPtnGearUp, m_nMaxGear);
                    bGearChange = true;
                }
                else if((tmpTargetSpeed<20.0)&&(tmpNowGear>2+m_nPtnGearUp)&&(2+m_nPtnGearUp<=m_nMaxGear)){
                    tmpNowGear = min(2 + m_nPtnGearUp, m_nMaxGear);
                    bGearChange = true;
                }
                else if((tmpTargetSpeed<40.0)&&(tmpNowGear>3+m_nPtnGearUp)&&(3+m_nPtnGearUp<=m_nMaxGear)){
                    tmpNowGear = min(3 + m_nPtnGearUp, m_nMaxGear);
                    bGearChange = true;
                }
                else if((tmpTargetSpeed<60.0)&&(tmpNowGear>4+m_nPtnGearUp)&&(4+m_nPtnGearUp<=m_nMaxGear)){
                    tmpNowGear = min(4 + m_nPtnGearUp, m_nMaxGear);
                    bGearChange = true;
                }
            }
            //--------------------------------
            // In case of acceleration previously
            //--------------------------------
            else{
```

```
if((tmpTargetSpeed>15.0)&&(tmpNowGear<2+m_nPtnGearUp)&&(2+m_nPtnGearUp<=m_nMaxGear)&&(2+m_nPtnGearUp>tmpPre
vGear)){
                    tmpNowGear = min(2 + m_nPtnGearUp, m_nMaxGear);
                    bGearChange = true;
                }
                else
if((tmpTargetSpeed>30.0)&&(tmpNowGear<3+m_nPtnGearUp)&&(3+m_nPtnGearUp<=m_nMaxGear)&&(3+m_nPtnGearUp>tmpPre
vGear)){
                    tmpNowGear = min(3 + m_nPtnGearUp, m_nMaxGear);
                    bGearChange = true;
                }
                else
if((tmpTargetSpeed>50.0)&&(tmpNowGear<4+m_nPtnGearUp)&&(4+m_nPtnGearUp<=m_nMaxGear)&&(4+m_nPtnGearUp>tmpPre
vGear)){
                    tmpNowGear = min(4 + m_nPtnGearUp, m_nMaxGear);
                    bGearChange = true;
                }
                else
if((tmpTargetSpeed>70.0)&&(tmpNowGear<5+m_nPtnGearUp)&&(5+m_nPtnGearUp<=m_nMaxGear)&&(5+m_nPtnGearUp>tmpPre
vGear)){
                    tmpNowGear = min(5 + m_nPtnGearUp, m_nMaxGear);
                    bGearChange = true;
                }
            }
        }

        //-------------------------------------------
        // Calculates current gear maintenance time
        //-------------------------------------------
        dRemainingTime = min((p_second->second.fTimes-p_tmpCalculate->second.fTimes)/10,
(double)GEAR_HOLD_TIME);
        dCurrentGearMaintainTime = GetGearMaintainTime( p_tmpCalculate, p_second, tmpNowGear,
dRemainingTime, bCurrentGearSpeedFollowed, bCurrentGearPatternFollowed, bCurrentGearChangeNeeded,
dDifferenceSpeed);

        if((p_previous->second.nFlag == ENGINE_DECELERATE)&&
               (tmpPrevGear != tmpNowGear )){
            if(dCurrentGearMaintainTime != min((double)GEAR_HOLD_TIME,dRemainingTime)){
                for(iContinueSettingGear = tmpNowGear; iContinueSettingGear <= m_nMaxGear;
iContinueSettingGear++ ){
                    dCurrentGearMaintainTime = GetGearMaintainTime( p_tmpCalculate, p_second,
iContinueSettingGear, dRemainingTime, bCurrentGearSpeedFollowed, bCurrentGearPatternFollowed,
bCurrentGearChangeNeeded, dDifferenceSpeed);
                    if( dCurrentGearMaintainTime == min((double)GEAR_HOLD_TIME,dRemainingTime) ){
                        break;
                    }
                }
                if( dCurrentGearMaintainTime == min((double)GEAR_HOLD_TIME,dRemainingTime) ){
                    tmpNowGear = iContinueSettingGear;
                }else{
                    tmpNowGear = p_previous->second.nCalcGear;
                    dCurrentGearMaintainTime = GetGearMaintainTime( p_tmpCalculate, p_second, tmpNowGear,
dRemainingTime, bCurrentGearSpeedFollowed, bCurrentGearPatternFollowed, bCurrentGearChangeNeeded,
dDifferenceSpeed);
                }
            }else if( bCurrentGearSpeedFollowed == false ){
                tmpNowGear = p_previous->second.nCalcGear;
                dCurrentGearMaintainTime = GetGearMaintainTime( p_tmpCalculate, p_second, tmpNowGear,
dRemainingTime, bCurrentGearSpeedFollowed, bCurrentGearPatternFollowed, bCurrentGearChangeNeeded,
dDifferenceSpeed);
                if( dDifferenceSpeed != 0 ){
                    bCurrentGearSpeedFollowed = false;
                }

            }
        }else if(( bGearChange == true )&&
               (dCurrentGearMaintainTime != min((double)GEAR_HOLD_TIME,dRemainingTime))){

            // shift-up calc.
            for(iContinueSettingGear = tmpNowGear; iContinueSettingGear <= m_nMaxGear;
iContinueSettingGear++ ){
                dCurrentGearMaintainTime = GetGearMaintainTime( p_tmpCalculate, p_second,
iContinueSettingGear, dRemainingTime, bCurrentGearSpeedFollowed, bCurrentGearPatternFollowed,
bCurrentGearChangeNeeded, dDifferenceSpeed);
```

```
                if( dCurrentGearMaintainTime == min((double)GEAR_HOLD_TIME,dRemainingTime) ){
                    break;
                }
            }
            if( dCurrentGearMaintainTime == min((double)GEAR_HOLD_TIME,dRemainingTime) ){
                tmpNowGear = iContinueSettingGear;
            }else{
                tmpNowGear = p_previous->second.nCalcGear;
                dCurrentGearMaintainTime = GetGearMaintainTime( p_tmpCalculate, p_second,
tmpNowChangePrevGear, dRemainingTime, bCurrentGearSpeedFollowed, bCurrentGearPatternFollowed,
bCurrentGearChangeNeeded, dDifferenceSpeed);
            }
            // hold gear calc.
            tmpNowChangePrevGear = p_previous->second.nCalcGear;
            dNowChangePrevGearMaintainTime = GetGearMaintainTime( p_tmpCalculate, p_second,
tmpNowChangePrevGear, dRemainingTime, bNowChangePrevGearSpeedFollowed, bNowChangePrevGearPatternFollowed,
bNowChangePrevGearChangeNeeded, dNowChangePrevDifferenceSpeed);

            if(( dNowChangePrevDifferenceSpeed < dDifferenceSpeed )&&
               ((bNowChangePrevGearChangeNeeded==true)&&(dNowChangePrevGearMaintainTime==0)!= true)){
                // hold now gear
                tmpNowGear = tmpNowChangePrevGear;
                dCurrentGearMaintainTime = dNowChangePrevGearMaintainTime;
                dDifferenceSpeed = dNowChangePrevDifferenceSpeed;

                p_tmpCalculate = RecordFixedGear(p_tmpCalculate, p_second, tmpNowGear,
dCurrentGearMaintainTime);
                continue;
            }
        }
        //------------------------------------------
        // If gear can be maintained during GEAR_HOLD_TIME, it is validated ...
        //------------------------------------------

if((bCurrentGearSpeedFollowed==true)&&(bCurrentGearPatternFollowed==true)&&(bCurrentGearChangeNeeded==false
)){
            p_tmpCalculate = RecordFixedGear(p_tmpCalculate, p_second, tmpNowGear, 1);
            continue;
        }

        //------------------------------------------
        // ... otherwise we look for the best gear
        //------------------------------------------
        else{
            if(( bCurrentGearPatternFollowed == false )&&(bCurrentGearChangeNeeded==false )){
                optimalGears = GetBestEngineSpeedMaintainGear(p_tmpCalculate, p_second, tmpNowGear,
dRemainingTime );
            }else if((bCurrentGearSpeedFollowed==false)&&
               (bCurrentGearChangeNeeded==false )&&(dCurrentGearMaintainTime ==
min((double)GEAR_HOLD_TIME,dRemainingTime))){

                if((tmpTargetSpeed<10.0)&&(tmpNowGear>1+m_nPtnGearUp)&&(1+m_nPtnGearUp<=m_nMaxGear)){
                    tmpNowGear = min(1 + m_nPtnGearUp, m_nMaxGear);
                }
                else if((tmpTargetSpeed<20.0)&&(tmpNowGear>2+m_nPtnGearUp)&&(2+m_nPtnGearUp<=m_nMaxGear)){
                    tmpNowGear = min(2 + m_nPtnGearUp, m_nMaxGear);
                }
                else if((tmpTargetSpeed<40.0)&&(tmpNowGear>3+m_nPtnGearUp)&&(3+m_nPtnGearUp<=m_nMaxGear)){
                    tmpNowGear = min(3 + m_nPtnGearUp, m_nMaxGear);
                }
                else if((tmpTargetSpeed<60.0)&&(tmpNowGear>4+m_nPtnGearUp)&&(4+m_nPtnGearUp<=m_nMaxGear)){
                    tmpNowGear = min(4 + m_nPtnGearUp, m_nMaxGear);
                }

                if( tmpNowGear -1 >= 2 ){
                    tmpNowGear--;
                }

                optimalGears = GetBestEngineSpeedMaintainGear(p_tmpCalculate, p_second, tmpNowGear,
dRemainingTime, dRemainingTime);
            }else{
                optimalGears = GetBestEngineSpeedMaintainGear(p_tmpCalculate, p_second, 2, dRemainingTime,
dRemainingTime);
            }
            if(optimalGears.iGearsNb !=0){
                bool bGearFound = false;
                // Look for 3Sec maintain + pattern follow + speed follow
                for(int i=0 ; i< optimalGears.iGearsNb ; i++){
                    if(optimalGears.bBestMaintainTime[i]){
                        tmpNowGear = optimalGears.iGearsID[i];
                        p_tmpCalculate = RecordFixedGear(p_tmpCalculate, p_second, tmpNowGear, 1);
                        bGearFound = true;
                        break;
                    }
                } if(bGearFound)    continue;

                // Look for 3Sec maintain
                for(int i=0 ; i< optimalGears.iGearsNb ; i++){
                    if(optimalGears.bBestMaintainTime[i]){
                        tmpNowGear = optimalGears.iGearsID[i];
                        p_tmpCalculate = RecordFixedGear(p_tmpCalculate, p_second, tmpNowGear, 1);
                        bGearFound = true;
                        break;
                    }
                } if(bGearFound)    continue;
            }
            else{
                tmpNowGear = 2;
                GetNe(tmpNowGear, tmpTargetSpeed ,tmpNe);
                if(tmpNe > m_fMaxOutputRotation){
                    while(tmpNowGear<m_nMaxGear)
                    {
                        tmpNowGear++;
                        GetNe(tmpNowGear, tmpTargetSpeed ,tmpNe);

                        if(tmpNe <= m_fMaxOutputRotation)  {
                            break;
                        }
                    };
                }
                p_tmpCalculate = RecordFixedGear(p_tmpCalculate, p_second, tmpNowGear, 1);
                continue;
            }
        }
        continue;
    }
    return true;
}
/**/
/*****************************************************************
 * Function name           : RecordFixedGear
 * Function summary        : With fixed gear, calculates speed and torque
 * Explanation             :
 *                         :
 * Argument (input)        : p_first       : First pointer of period to be calculated
 * Argument (input)        : p_second      : Last pointer of period to be calculated
 * Argument (input)        : iGear         : Fixed gear that has to be used during calcul
 * Argument (input)        : dMaintainTime : Time during which calcul has to be continued
 * Argument (output)       : None
 * Argument (I/O)          : None
 * Return value            : Last pointer having been calculated
 * Created by              :
 * Updated on (created on) :
 * Remarks                 :
 *****************************************************************/
map<double,stCalculateData>::iterator TCalculateProc::RecordFixedGear(map<double,stCalculateData>::iterator
p_start,
                                     map<double,stCalculateData>::iterator p_end,
                                    int iGear,
                                    double dMaintainTime)
{
    map<double,stCalculateData>::iterator p_tmpCalculate;      // Temporary pointer
    map<double,stCalculateData>::iterator p_previous;          // Previous data

    int tmpNowGear = iGear;                                    // Temporary gear
    int tmpPrevGear;                                           // Previous gear

    double tmpNe;                                              // Calculated engine speed

    double fCarAcc;                                            // Acceleration value (used to calculate
torque)
    double tmpTe;                                             // Calculated torque
    double tmpTeMax;                                          // Maximum torque

    double tmpTargetSpeed;                                    // Calculated speed

    double tmpPrevVAna_sp;                                    // Previous analysis speed
    double tmpPrevGearTime;                                   // Gear required time
    double tmpPrevCalcTime;                                   // Required time

    for(p_tmpCalculate=p_start ; p_tmpCalculate!=p_end &&
((p_tmpCalculate->second.fTimes-p_start->second.fTimes)/10<dMaintainTime); p_tmpCalculate++)
    {
        tmpTargetSpeed = p_tmpCalculate->second.fVTarget_sp;

        p_previous = p_tmpCalculate;
        p_previous--;

        tmpPrevGearTime = p_previous->second.nGearTime;       // Previous gear time
        tmpPrevVAna_sp  = p_previous->second.fVAna_sp;        // Previous analysis speed
        tmpPrevCalcTime = p_previous->second.nCalcTime;       // Previous required time
        tmpPrevGear     = p_previous->second.nCalcGear;

        //---------------------
        // Engine speed calcul
        //---------------------
        GetNe(tmpNowGear, tmpTargetSpeed ,tmpNe);
        if( tmpNe<m_fClutch_MeetNe && (p_previous->second.fVAna_sp==0 ||
p_previous->second.bClutchMeetMode==true))
        {
            tmpNe=m_fClutch_MeetNe;
            p_tmpCalculate->second.bClutchMeetMode=true;
        }
        if( tmpNe > m_fMaxOutputRotation){
            if(tmpNowGear>=m_nMaxGear){
                tmpNe=m_fMaxOutputRotation; //-> Engine speed is limited to its maximum
                GetV(tmpNowGear, m_fMaxOutputRotation, tmpTargetSpeed);
            }
        }

        //---------------------
        // Torque calcul
        //---------------------
        fCarAcc = ((tmpTargetSpeed - tmpPrevVAna_sp)/(tmpPrevCalcTime/10.0))/3.6;
        GetTe(tmpNowGear,tmpTargetSpeed,fCarAcc, tmpTe);
        tmpTeMax = GetLineReviseMaxTorque(tmpNe);
        if(tmpTe>tmpTeMax){// Torque is exceeded: speed cannot be followed, it is optimized
            CalcTeMaxSp(tmpNowGear, tmpPrevCalcTime, tmpPrevVAna_sp, tmpTargetSpeed, tmpNe, tmpTe );
        }

        //----------------
        // Saving results
        //----------------
        if(tmpNowGear==tmpPrevGear){    // If gear has not been changed: gearTime is incremented
            p_tmpCalculate->second.nGearTime= p_previous->second.nGearTime + p_previous->second.nCalcTime;
        }else{  // If gear has been changed: gearTime is reseted
            p_tmpCalculate->second.nGearTime=p_previous->second.nCalcTime;
        }
        p_tmpCalculate->second.nCalcGear = tmpNowGear;
        p_tmpCalculate->second.fVAna_sp = tmpTargetSpeed;
        p_tmpCalculate->second.fNeRevo  = tmpNe;
        p_tmpCalculate->second.fTe      = tmpTe;
    }

    if(p_tmpCalculate!=p_start) p_tmpCalculate--;

    return p_tmpCalculate;
}

/**/
/***************************************************************
 * Function name          : Calculate_Engine_DECELERATE
 * Function summary       : In case of deceleration, look for correct clutch meet and speed
 * Explanation            :
 *                        :
```

```
 * Argument (input)        : p_first   : First pointer
 * Argument (input)        : p_second  : Next setting pointer
 * Argument (output)       : None
 * Argument (I/O)          : None
 * Return value            : true : Normal    false : Failure
 * Created by              :
 * Updated on (created on) :
 * Remarks                 :
 ****************************************************************/
bool TCalculateProc::Calculate_Engine_DECELERATE(map<double,stCalculateData>::iterator p_first,
                        map<double,stCalculateData>::iterator p_second )
{
    map<double,stCalculateData>::iterator p_tmpCalculate;        // Temporary pointer
    map<double,stCalculateData>::iterator p_previous;            // Previous data
    map<double,stCalculateData>::iterator p_next;                // Next data

    int prevGear;                                                // Temporary gear
    double currentTargetSpeed;                                   // Previous analysis speed
    double prevVana_sp;
    double tmpVana_sp;
    double tmpCarAcc;                                            // Previous analysis speed
    double tmpTe;
    double tmpNe;

    for(p_tmpCalculate=p_first ; p_tmpCalculate!=p_second ; p_tmpCalculate++){
        currentTargetSpeed = p_tmpCalculate->second.fVTarget_sp;

        p_previous = p_tmpCalculate;
        p_previous--;
        prevVana_sp = p_previous->second.fVAna_sp;
        prevGear    = p_previous->second.nCalcGear;

        //--------------------------------
        // Regulating the car speed
        //--------------------------------
        if(prevVana_sp < currentTargetSpeed){                    // If wanted speed has not been reached
yet
            p_next=p_tmpCalculate;
            p_next++;
            p_tmpCalculate->second.nFlag = ENGINE_ACCELERATE;
            Calculate_Engine_ACCELERATE(p_tmpCalculate, p_next);

            tmpVana_sp = p_tmpCalculate->second.fVAna_sp ;
            continue;
        }
        else {
            tmpVana_sp = currentTargetSpeed;
        }

        //--------------------------------
        // Regulating the engine speed
        //--------------------------------
        if(prevGear!=0){
            GetNe(prevGear, tmpVana_sp ,tmpNe);
            if((( prevGear == 1 + m_nPtnGearUp)&&( 1 + m_nPtnGearUp<= m_nMaxGear )&&( currentTargetSpeed <
5.0 ))||
               (( prevGear == 2 + m_nPtnGearUp)&&( 2 + m_nPtnGearUp<= m_nMaxGear )&&( currentTargetSpeed <
10.0 ))||
               (( prevGear == 3 + m_nPtnGearUp)&&( 3 + m_nPtnGearUp<= m_nMaxGear )&&( currentTargetSpeed <
15.0 ))||
               (( prevGear == 4 + m_nPtnGearUp)&&( 4 + m_nPtnGearUp<= m_nMaxGear )&&( currentTargetSpeed <
20.0 ))||
               (( prevGear >= 5 + m_nPtnGearUp)&&( 5 + m_nPtnGearUp<= m_nMaxGear )&&( currentTargetSpeed <
30.0 ))){
                    tmpNe=m_fIdleSpeed;
                    prevGear=0;
               }
        }

        if(prevGear==0){
            tmpTe=0;
            tmpNe=m_fIdleSpeed;
        }
        else {
            // Calculate acceleration as VAna speed may not be the same as VTarget speed
            tmpCarAcc = (tmpVana_sp - prevVana_sp) / (p_tmpCalculate->second.nCalcTime/10) /3.6;
            GetTe(prevGear, tmpVana_sp, tmpCarAcc, tmpTe);

            //Torque cannot exceed its maximum value
            tmpTe=min(GetLineReviseMaxTorque(tmpNe), tmpTe);
        }
        //--------------------------------
        // Saving calculated values
        //--------------------------------
        p_tmpCalculate->second.nCalcGear   = prevGear;
        p_tmpCalculate->second.fNeRevo = tmpNe;
        p_tmpCalculate->second.fTe     = tmpTe;
        p_tmpCalculate->second.fVAna_sp= tmpVana_sp;

        if(prevGear != p_previous->second.nCalcGear)
            p_tmpCalculate->second.nGearTime=p_previous->second.nCalcTime;
        else
            p_tmpCalculate->second.nGearTime=p_previous->second.nGearTime + p_previous->second.nCalcTime;
    }
    return true;
}


/**/
/****************************************************************
 * Function name           : GetGearPass
 * Function summary        : Obtains gear transmission efficiency
 * Explanation             : Obtains gear transmission efficiency.
 *                         :
 * Argument (input)        : nGear: gear
 * Argument (output)       : None
 * Argument (I/O)          : None
 * Return value            : double Transmission efficiency
 * Created by              :
 * Updated on (created on) :
 * Remarks                 :
 ****************************************************************/
double TCalculateProc::GetGearPass( int nGear )
{
    //--------------------------------
    // Set transmission efficiency based on gear ratio.
    //--------------------------------
    if( m_vGearRatio[nGear-1] == 1 ){     // If gear ratio is 1:0
        return DEF_FORCE_ON98;
    }else{
        return DEF_FORCE_OFF95;
    }
}
/**/
/****************************************************************
 * Function name           : GetLineReviseMaxTorque
 * Function summary        : Max. torque data interpolation processing
 * Explanation             : Proportionaly to Torque engine specifications,
 *                         : the torque corresponding to given revolution (fNe) is calculated with a linear
approximation.
 *                         :
 * Argument (input)        : fNe: Revolution
 * Argument (output)       : None
 * Argument (I/O)          : None
 * Return value            : double Torque
 * Created by              :
 * Updated on (created on) :
 * Remarks                 :
 ****************************************************************/
double  TCalculateProc::GetLineReviseMaxTorque(double fNe)
{
    double  fNeA,fNeB,fTorqueA,fTorqueB,fMaxTorque;
    stTorqueData  tmpMaxTorque;                      // Temporary max. torque data

    if (m_MaxTorque.empty()){                        // Return 0 if there is no max. torque data
        return 0.0;
    }

    //--------------------------------
    // Find appropriate revolution and max. loss torque data from array.
    //---------------------------------
    memset( &tmpMaxTorque, 0x00, sizeof( tmpMaxTorque));
    tmpMaxTorque.d_EngineRevolutions = fNe;
    p_MaxTorque = m_MaxTorque.lower_bound( tmpMaxTorque);

    if( p_MaxTorque == m_MaxTorque.end() ){          //If pointer reaches the last value, that value is
returned
        p_MaxTorque = m_MaxTorque.end();
        p_MaxTorque--;
        return p_MaxTorque->d_EngineTorque;
    }

    fNeB = p_MaxTorque->d_EngineRevolutions;
    fTorqueB = p_MaxTorque->d_EngineTorque;

    //---------------------------------
    // Obtain preceding data.
    //---------------------------------
    if( p_MaxTorque != m_MaxTorque.begin() ){
        p_MaxTorque--;
        fTorqueA = p_MaxTorque->d_EngineTorque;
        fNeA = p_MaxTorque->d_EngineRevolutions;
    }

    //---------------------------------
    // Next data if first data.
    //---------------------------------
    else{
        fTorqueA = p_MaxTorque->d_EngineTorque;
        fNeA = p_MaxTorque->d_EngineRevolutions;
        p_MaxTorque++;
        fNeB = p_MaxTorque->d_EngineRevolutions;
        fTorqueB = p_MaxTorque->d_EngineTorque;
    }


    if ((fNeB - fNeA) == 0) return 0;                //Prevent dividing by 0.

    //---------------------------------
    //Obtain appropriate max. torque by linear interpolation (polygonal line).
    //---------------------------------
    fMaxTorque = fTorqueA +
            (fTorqueB - fTorqueA) /
            (fNeB - fNeA) *
            (fNe - fNeA);

    return fMaxTorque;
}
/**/
/*****************************************************************
 * Function name           : CalcTeMaxSp
 * Function summary        : Target-speed follow calculation processing
 * Explanation             : When speed changes from A to B,
 *                         : speed fV is calculated if target-speed follow is impossible in time fTm.
 * Argument (input)        : nGear  : Gear to be used
 * Argument (input)        : fTm    : Usage time
 * Argument (input)        : fPrevV : Previous speed
 * Argument (I/O)          : fNe    : Engine speed
 * Argument (I/O)          : fV     : Speed for this time/speed after re-calculation
 * Argument (I/O)          : fTe    : Torque for this time/speed after re-calculation
 * Return value            : true : Converged  ;  false : Not converged
 * Created by              :
 * Updated on (created on) :
 * Remarks                 :
 *****************************************************************/
bool TCalculateProc::CalcTeMaxSp(int nGear, double fTm, double fPrevV, double &fV, double &fNe, double &fTe
)
{
    double diff;
    double engineTorque;
    double maxTorque;
    double carAcc;

    double ds=1;
    int flag=0;
    int ret;

    double tmpNe;

    //--------------------------------
    // If engine torque is already less than maximum torque, calculation loop is stoped
    //--------------------------------
    if( fNe < m_fClutch_MeetNe ){
        fNe = m_fClutch_MeetNe;
    }
    if( fNe >= m_fMaxOutputRotation ){
        fNe = m_fMaxOutputRotation;


        ret = GetV( nGear, fNe, fV );
        if( ret == NG ){
            return false;}
    }

    carAcc = (( fV - fPrevV) / (fTm/10)) /3.6;           // Calculate acceleration

    ret=GetTe(nGear,fV, carAcc, engineTorque);           // Calculate Torque for fV speed
    if( ret == NG ){
        return false;
    }
    maxTorque = GetLineReviseMaxTorque (fNe);            // Calculating maximum torque value
    diff= maxTorque-engineTorque;
    if(diff>0) return true;                              // loop is stoped if torque is less than maximum
    else fV-=ds;


    //--------------------------------
    // If torque exceeds maximum a better value is calculated by approximation
    //--------------------------------
    while(flag==0 && ds!=0){
        ret = GetNe( nGear, fV, fNe);                    // Calculate engine speed having fV as car speed
        if( ret == NG ){
            return false;
        }
        if( fNe < m_fClutch_MeetNe ){
            fNe = m_fClutch_MeetNe;
        }
        if( fNe >= m_fMaxOutputRotation ){
            fNe = m_fMaxOutputRotation;

            ret = GetV( nGear, fNe, fV );
            if( ret == NG ){
                return false;
            }
        }

        carAcc = (( fV - fPrevV) / (fTm/10)) /3.6;       // Calculate acceleration

        ret=GetTe(nGear,fV, carAcc, engineTorque);       // Calculate Torque for fV speed
        if( ret == NG ){
            return false;
        }
        maxTorque = GetLineReviseMaxTorque (fNe);        // Linear interpolation
        diff= maxTorque-engineTorque;

        if( 0<=diff && diff<1.0E-6){
            flag=1;
        }
        else{
            if(diff<0) fV-=ds;
            else{
                ds=ds/2;
                fV+=ds;
            }
        }
    }

    fTe=engineTorque;

    return( true );
```

```
}

/**/
/*****************************************************************
 * Function name           : calcRL
 * Function summary        : Rolling resistance calculation processing
 * Explanation             : Rolling resistance is calculated.
 *                         :
 * Argument (input)        : fcarSpeed  : Vehicle speed
 * Argument (output)       : None
 * Argument (I/O)          : None
 * Return value            : double Rolling resistance value
 * Created by              :
 * Updated on (created on) :
 * Remarks                 :
 *****************************************************************/
double  TCalculateProc::calcRL(double fcarSpeed)
{
    double fCarWeight;
    double fRL;

    fCarWeight = GetCarWeight(false);   //Read and calculate weight data.

    fRL = ((double)((0.00513 + 17.6/fCarWeight) * fCarWeight)) +
            ((double)((0.00299 * m_fOverWidth * m_fOverHeight - 0.000832)) * (fcarSpeed * fcarSpeed));
    return fRL;
}
/**/
/*****************************************************************
 * Function name           : GetCarWeight
 * Function summary        : Curb vehicle weight data calculation processing
 * Explanation             : Curb vehicle weight is calculated.
 *                         :
 * Argument (input)        : bFlag : If true, equivalent rotational inertia mass ratio is included, and not
included if false.
 * Argument (output)       : None
 * Argument (I/O)          : None
 * Return value            : double Curb vehicle weight value
 * Created by              :
 * Updated on (created on) :
 * Remarks                 :
 *****************************************************************/
double  TCalculateProc::GetCarWeight(bool bFlag, int nGear)
{
    double fCarWeight;
    double fGearRatio;

    if( bFlag ){
        fGearRatio=m_vGearRatio[nGear-1];
        fCarWeight=m_fCarIniW + m_fCarIniW*M_FACT + m_fCarIniW*E_FACT* fGearRatio*fGearRatio +
m_fCarPayload + PERSON_W;
    }else{
        fCarWeight=m_fCarPayload + m_fCarIniW + PERSON_W;
    }
    return fCarWeight;
}
/**/
/*****************************************************************
 * Function name           : GetNe
 * Function summary        : Returns the number of rotations per minute in (rpm)
 * Explanation             : Revolution's speed is calculated
 *                         :
 * Argument (input)        : nGear             : gear
 * Argument (input)        : fVg               : Vehicle speed (Km/h)
 * Argument (output)       : fNe               : Revolution 's speed (rpm)
 * Argument (I/O)          : None
 * Return value            : OK  : Normal   NG: Failure
 * Created by              :
 * Updated on (created on) :
 * Remarks                 :
 *****************************************************************/
int TCalculateProc::GetNe(int nGear,double fVg,double &fNe)
{
    double fGearBoxRatio;
    fGearBoxRatio = m_vGearRatio[nGear-1] * m_fLastReduceGear;

    fNe = fVg / 60.0 * fGearBoxRatio * 1000.0 / (2.0 * PI * m_fTireRollRadius);
    return OK;
}
/**/
/*****************************************************************
 * Function name          : GetV
 * Function summary       : Speed calculation processing
 * Explanation            : Speed is calculated.
 *                        :
 * Argument (input)       : nGear             : gear
 * Argument (input)       : fNe               : Revolution
 * Argument (output)      : fVg               : Vehicle speed (Vg)
 * Argument (I/O)         : None
 * Return value           : OK  : Normal   NG: Failure
 * Created by             :
 * Updated on (created on) :
 * Remarks                :
 *****************************************************************/
int TCalculateProc::GetV( int nGear,double fNe,double &fVg)
{
    double fGearBoxRatio;

    fGearBoxRatio = m_vGearRatio[nGear-1] * m_fLastReduceGear;

    fVg = fNe * 60.0 / fGearBoxRatio / 1000.0 * (2.0 * PI * m_fTireRollRadius);
    return OK;
}
/**/
/*****************************************************************
 * Function name          : GetTe
 * Function summary       : Torque calculation processing
 * Explanation            : Torque is calculated and interpolation data is considered if needed (see
bApplyCorrection)
 * Argument (input)       : nGear             : gear
 * Argument (input)       : fV                : Vehicle speed (fV)
 * Argument (input)       : fA                : Acceleration for fV
 * Argument (input)       : fNe               : Revolution
 * Argument (input)       : bMaxLimit         : A max limit is applied if necessary
 * Argument (output)      : fTe               : Torque
 * Argument (I/O)         : None
 * Return value           : OK  : Normal   NG: Failure
 * Created by             :
 * Updated on (created on) :
 * Remarks                :
 *****************************************************************/
int TCalculateProc::GetTe( int nGear,double fTargetSp,double fCarAcc, double &fTe)
{
    double fnGearPass;                     // n'th-gear ratio (transmission efficiency) data
    double fCarMt;                         // Vehicle body weight
    double fGearBoxRatio,fRL;
    double maxTe;


    fCarMt = GetCarWeight(true,nGear);      // Vehicle body weight
    fnGearPass = GetGearPass(nGear);        // Obtain gear transmission efficiency.

    fGearBoxRatio = m_vGearRatio[nGear-1] * m_fLastReduceGear;

    fRL = calcRL(fTargetSp);

    //Engine torque is calculated here
    fTe = ((G*m_fTireRollRadius)/( fGearBoxRatio * fnGearPass * UD))*( fRL + (fCarMt / G) * fCarAcc );

    return OK;
}
/**/
/*****************************************************************
 * Function name          : DispCalculateData
 * Function summary       : Processing for parameter display during processing
 * Explanation            : Specification data from read file is
 *                        : displayed on screen
 * Argument (input)       : None
 * Argument (output)      : None
 * Argument (I/O)           : None
 * Return value             :
 * Created by               :
 * Updated on (created on)  :
 * Remarks                  :
 ****************************************************************/
void    TCalculateProc::DispCalculateData(void)
{
    char buf[256];
    double fCarM;
    double fDW;
    double fGearvalue;

    fCarM = GetCarWeight(false);

    // Conversion infomation
    cout << "Ver " << MY_VERSION << endl;

    sprintf( buf, " mass  =%8.2f[kg]¥n", fCarM );
    cout << buf;
    sprintf( buf, " WO    =%8.2f[kg], Wtest =%8.2f[kg]¥n", m_fCarIniW , fCarM );
    cout << buf;
    sprintf( buf, " Width =%8.3f[m],  Height=%8.3f[m], Tire radius=%8.3f[m]¥n",
                                m_fOverWidth,
                                m_fOverHeight,
                                m_fTireRollRadius );
    cout << buf;
    sprintf( buf, " Crew  =%3d¥n", (int)(m_fPersons) );
    cout << buf;
    sprintf( buf, "¥n" );
    cout << buf;
    sprintf( buf, " Nidle =%8.2f[rpm], Nex  =%8.2f[rpm]¥n",
                                m_fIdleSpeed,
                                m_fMaxOutputRotation );
    cout << buf;
    sprintf( buf, " Nes   =%8.2f[rpm]¥n",
                                m_fClutch_MeetNe );
    cout << buf;
    sprintf( buf, " MuAir =%10.6f [kgf/(km/h)^2], MuRoll =%10.6f [kgf/kg]¥n",
                    (0.00299 * m_fOverWidth * m_fOverHeight - 0.000832),
                    (0.00513 + 17.6/fCarM) );
    cout << buf;
    sprintf( buf, "¥n" );
    cout << buf;
    sprintf( buf, " Number of gear = %2d¥n", m_nMaxGear );
    cout << buf;
    sprintf( buf, " gear   ratio  efficiency  DW[kg]¥n");
    cout << buf;

    for( int  gear = 1; gear <= m_nMaxGear; gear++ ){
       fDW = (M_FACT + E_FACT * m_vGearRatio[gear-1] * m_vGearRatio[gear-1]) * m_fCarIniW;
       sprintf( buf, " %3d:   %6.3f    %6.3f  %12.5f ¥n",
                gear,
                fGearvalue,
                GetGearPass(gear),
                fDW );

       cout << buf;
    }
    sprintf( buf, " fin:   %6.3f    %6.3f¥n", m_fLastReduceGear, UD );
    cout << buf;
    sprintf( buf, "¥n" );
    cout << buf;

}
/**/
/****************************************************************
 * Function name            : WriteAllCalculateData
 * Function summary         : Processed data output processing
 * Explanation              : Processing result is output to file
 *                          :
 * Argument (input)         : None
 * Argument (output)        : None
 * Argument (I/O)           : None
 * Return value             : OK  : Normal   NG: Failure
 * Created by              :
 * Updated on (created on) :
 * Remarks                 :
 ***************************************************************/
int TCalculateProc::WriteAllCalculateData()
{
    int    nRet;
    char   buf[1024];
    FILE   *m_pFile;
    double fMaxTe;
    bool   tmpbTe_f;
    //bool   tmpbN_norm_f;
    bool   tmpbT_norm_f;

    double tmpfVref;
    double tmpfVana;
    double tmpfNe;
    double tmpfTe;
    double tmpN_norm;
    double tmpT_norm;
    char   tmp_strfTe[128];
    char   tmp_strfNe[128];
    char   tmp_strN_norm[128];
    char   tmp_strT_norm[128];


    if( m_sOutputFileName == "" ){                          //Looking for output file name
        cerr << "Please enter an output file name: ";
        cin >> m_sOutputFileName;
    }

    if( ( m_pFile = fopen( m_sOutputFileName.c_str(), "wt" ) ) == NULL ){
        sprintf( buf, "%s¥n¥nThe file is not found.", m_sOutputFileName.c_str() );
        cout << buf << endl;
        return NG;
    }

    nRet = WriteHead(m_pFile);
    if (nRet != OK){
        return NG;
    }

    int lineNB=0;

    //----------------------------------
    // Loops over all analysis data items:
    //----------------------------------

for(p_setCalculateData=setCalculateData.begin();p_setCalculateData!=setCalculateData.end();p_setCalculateDa
ta++){

        fMaxTe = GetLineReviseMaxTorque(p_setCalculateData->second.fNeRevo);
        tmpfTe  = p_setCalculateData->second.fTe;
        tmpN_norm = (((p_setCalculateData->second.fNeRevo - m_fIdleSpeed)/( m_fMaxOutputRotation -
m_fIdleSpeed )) * 100.0);
        tmpT_norm = ((p_setCalculateData->second.fTe / fMaxTe) * 100.0);

        if(tmpN_norm>100) tmpN_norm=99999;
        if(tmpT_norm>100) tmpN_norm=99999;


        tmpbTe_f = false;
        //tmpbN_norm_f = false;
        tmpbT_norm_f = false;

        if( tmpfTe < 0.0 ){
            tmpbTe_f = true;
        }
        /*if( tmpN_norm < 0.0 ){
            tmpbN_norm_f = true;
        }*/
        if( tmpT_norm < 0.0 ){
            tmpbT_norm_f = true;
        }
        tmpfVref = p_setCalculateData->second.fVTarget_sp;
        tmpfVana = p_setCalculateData->second.fVAna_sp;
        tmpfNe  = p_setCalculateData->second.fNeRevo;

        if( tmpbTe_f == false ){
            sprintf( tmp_strfTe, "%.1f", tmpfTe );
        }else{
            sprintf( tmp_strfTe, "%s", "M" );
        }
        sprintf( tmp_strfNe, "%.1f", tmpfNe );

        sprintf( tmp_strN_norm, "%.2f", tmpN_norm );


        if( tmpbT_norm_f == false ){
            sprintf( tmp_strT_norm, "%.2f", tmpT_norm );
        }else{
            sprintf( tmp_strT_norm, "%s", "M" );
        }

        lineNB++;

        int tmpAccumTime = (int)(p_setCalculateData->second.fTimes /10);

        sprintf(buf,"%d¥t%.2f¥t%.2f¥t%s¥t%s¥t%s¥t%s¥t%d",
                tmpAccumTime,                             // Accumulated time
                tmpfVref,                                 // Reference vehicle speed
                tmpfVana,                                 // Analysis vehicle speed
                tmp_strfNe,                               // Engine speed
                tmp_strfTe,                               // Engine torque
                tmp_strN_norm,
                tmp_strT_norm,
                p_setCalculateData->second.nCalcGear );       // Gear

        nRet = fprintf(m_pFile, "%s¥n", buf);
        if (nRet == EOF){
            fclose(m_pFile);
            cout << MSG_WRITE_FILE_ERROR << endl;
            return NG;
        }
    }

    fclose(m_pFile);
    return OK;
}

/**/
/*****************************************************************
 * Function name          : WriteHead
 * Function summary       : Analysis data header output processing
 * Explanation            : Header is output to processing result file
 *                        :
 * Argument (input)       : *fp  : Analysis data output file name
 * Argument (output)      : None
 * Argument (I/O)         : None
 * Return value           :  OK  : Normal   NG: Failure
 * Created by             :
 * Updated on (created on) :
 * Remarks                :
 *****************************************************************/
int TCalculateProc::WriteHead(FILE *fp)
{
    int nRet;
    string  szFieldTitle;

    szFieldTitle = DEF_PRINT_POS1;
    szFieldTitle = szFieldTitle + "¥t" + DEF_PRINT_POS2;
    szFieldTitle = szFieldTitle + "¥t" + DEF_PRINT_POS3;
    szFieldTitle = szFieldTitle + "¥t" + DEF_PRINT_POS4;
    szFieldTitle = szFieldTitle + "¥t" + DEF_PRINT_POS5;
    szFieldTitle = szFieldTitle + "¥t" + DEF_PRINT_POS6;
    szFieldTitle = szFieldTitle + "¥t" + DEF_PRINT_POS7;
    szFieldTitle = szFieldTitle + "¥t" + DEF_PRINT_POS8;

    nRet = fprintf(fp, "%s¥n", szFieldTitle.c_str());
    if (nRet == EOF){
        cout << MSG_WRITE_FILE_ERROR << endl;
        return NG;
    }

    return OK;
}

/**/
/****************************************************************
 * Function name           : Data_Acquisition
 * Function summary        : Check the presence of input data and launch their copy into parameters
 * Explanation             : Reads the MAIN_ENVFILE, check its presence and save its content.
 *                         :
 * Argument (input)        : None
 * Argument (output)       : None
 * Argument (I/O)          : None
 * Return value            : 1: success ; others:failure  (an error code is returned)
 * Created by              :
 * Updated on (created on) :
 * Remarks                 :
 ****************************************************************/
int TCalculateProc::Data_Acquisition()
{
     //--------------------------------
     //Locale parameters declaration
     //--------------------------------
     FILE *fp_MainEnvfile;                     // Pointer to the main environment file
     char tmp_lineRead[LINE_MAX_LENGTH];
     string s_dataFileNames[DATA_FILES_NUMBER];// Array of strings containing data file names
     int i_retValue=OK;


     //--------------------------------
     // Opens and Reads the Main_Envfile
     //--------------------------------
     fp_MainEnvfile = fopen( m_sInputFileName.c_str(), "r" );
     if((fp_MainEnvfile == NULL)||(ferror(fp_MainEnvfile)))
        return ERROR_MAIN_FILE_NOT_FOUND;


     //--------------------------------
     //Retrieve data files names from Main_Envfile file
     //--------------------------------
     int nbFile=0;

     while(fgets(tmp_lineRead, LINE_MAX_LENGTH, fp_MainEnvfile)!=NULL && nbFile!=DATA_FILES_NUMBER ){
        strtok(tmp_lineRead, " ¥r¥t¥n"); // Stops the name when " ", "¥r", "¥t" or"¥n" caracter is
encountered
        s_dataFileNames[nbFile]=string(tmp_lineRead);
        nbFile++;
     }

     //--------------------------------
     //Saving Environment datas
     //--------------------------------
     i_retValue=SaveEnvironmentData(s_dataFileNames[0]);
     if(i_retValue!=OK) return i_retValue;

     //--------------------------------
     //Saving Specification datas
     //--------------------------------
     i_retValue=SaveSpecificationData(s_dataFileNames[1]);
     if(i_retValue!=OK) return i_retValue;

     //--------------------------------
     //Saving Torque datas
     //--------------------------------
     i_retValue=SaveTorqueData(s_dataFileNames[2]);
     if(i_retValue!=OK) return i_retValue;

    return i_retValue;
}
/**/
/****************************************************************
```

```
 * Function name           : SaveEnvironmentData
 * Function summary        : Copies environment datas in locale vector
 * Explanation             :
 *                         :
 * Argument (input)        : None
 * Argument (output)       : None
 * Argument (I/O)          : None
 * Return value            : 1: success ; others:failure, an error code is returned
 * Created by              :
 * Updated on (created on) :
 * Remarks                 :
 ****************************************************************/
int TCalculateProc::SaveEnvironmentData(string fileName)
{
    FILE *fp_Envfile;
    stCalculateData tmpCalculateData;

    char *p;
    char tmp_lineRead[LINE_MAX_LENGTH];
    memset(tmp_lineRead, 0x00, LINE_MAX_LENGTH ); //Reset of tmp_lineRead

    //--------------------------------
    //Opens and Reads the Envfile
    //--------------------------------
    fp_Envfile = fopen(fileName.c_str() , "r" );
    if((fp_Envfile == NULL)||(ferror(fp_Envfile)))
        return ERROR_ENV_FILE_NOT_FOUND;

    //--------------------------------
    //Reading the file and storing elements in structure
    //--------------------------------
    for(int row=0; fgets(tmp_lineRead, LINE_MAX_LENGTH, fp_Envfile); row++){
        if(row==0) continue;      //We skip the header line

        //Reset of "CalculateData" structure
        memset(&tmpCalculateData, 0x00, sizeof(tmpCalculateData));

        //Time
        p = strtok(tmp_lineRead, " ¥t,;%¥n");
        if( p == NULL ) continue;// Incomplete or empty line is ignored
        tmpCalculateData.fTimes= atof(p) * 10; // Sets accumulated seconds in msec.

        //Speed
        p = strtok(NULL, " ¥t,;%¥n");
        if( p == NULL ) continue;// Incomplete or empty line is ignored
        tmpCalculateData.fVTarget_sp=atof(p);

        //Shift
        p = strtok(NULL, " ¥t,;%¥n");
        if( p == NULL ) continue;// Incomplete or empty line is ignored
                tmpCalculateData.nCalcGear=atoi(p);

        tmpCalculateData.bClutchMeetMode=false;

        //Read datas are saved in array
        setCalculateData.insert(pair<double,stCalculateData>(tmpCalculateData.fTimes,tmpCalculateData));
    }
    fclose(fp_Envfile);
    if(setCalculateData.empty()) return ERROR_ENV_FILE_EMPTY;


    //--------------------------------
    // Updates all section time by calculating 'CalcTime' value
    //--------------------------------
    map<double,stCalculateData>::iterator p_tmp;
    map<double,stCalculateData>::iterator p_next;

    for( p_tmp = setCalculateData.begin() ; p_tmp != setCalculateData.end() ; p_tmp++ ){
        p_next = p_tmp; p_next++;
        if( p_next != setCalculateData.end() ){
            p_tmp->second.nCalcTime = (int)(p_next->second.fTimes - p_tmp->second.fTimes); // Sets section
using accumulated time
        }
    }
    return OK;
}
/**/
/****************************************************************
 * Function name          : SaveSpecificationData
 * Function summary       : Copies specification datas in locale vector
 * Explanation            :
 *                        :
 * Argument (input)       : None
 * Argument (output)      : None
 * Argument (I/O)         : None
 * Return value           : 1: success ; others:failure, an error code is returned
 * Created by             :
 * Updated on (created on) :
 * Remarks                :
 ****************************************************************/
int TCalculateProc::SaveSpecificationData(string fileName)
{
    FILE *fp_Specfile;
    char *p;
    char tmp_lineRead[LINE_MAX_LENGTH];
    memset(tmp_lineRead, 0x00, LINE_MAX_LENGTH ); //Reset of tmp_lineRead
    double d_tmpSpecValue;


    //--------------------------------
    //Opens and Reads the Envfile
    //--------------------------------
    fp_Specfile = fopen(fileName.c_str() , "r" );
    if((fp_Specfile == NULL)||(ferror(fp_Specfile)))
        return ERROR_SPEC_FILE_NOT_FOUND;

    //--------------------------------
    //Reading the file and storing the specifications value
    //--------------------------------
    for(int row=0; fgets(tmp_lineRead, LINE_MAX_LENGTH, fp_Specfile); row++){
        //If not number, line is not read
        if(((int)tmp_lineRead[0] < 0x30) || ((int)tmp_lineRead[0])> 0x39)
                continue;

        p = strtok(tmp_lineRead, " ¥t¥n");
        d_tmpSpecValue=atof(p);

        //--------------------------------
        //Testing default values
        //--------------------------------
        if(p == NULL){
                if(row == 0) return ERROR_SPEC_DATA_FORMAT;                     //Curb Vehicule weight
           else if(row == 1) return ERROR_SPEC_DATA_FORMAT;                     //Max payload
           else if(row == 2) d_tmpSpecValue=0;                                  //Number of persons
           else if(row == 3) d_tmpSpecValue=0;                                  //Overall vehicule weight
           else if(row == 4) d_tmpSpecValue=0;                                  //Overall vehicule width
           else if(row == 5) d_tmpSpecValue=0;                                  //Tire dynamic rolling
radius
           else if(row == 6) d_tmpSpecValue=DEF_MAXGEAR;                        //Number of gear positions
           else if(row <= 6+m_vSpecificationData[6]) d_tmpSpecValue=DEF_GEAR_RATIO;         //Gear ratio
           else if(row == 7+m_vSpecificationData[6]) d_tmpSpecValue=DEF_FINAL_REDUC_RATIO;  //Final
reduction ration
           else if(row == 8+m_vSpecificationData[6]) d_tmpSpecValue=DEF_IDLING_ENGINE_SPEED;//Idling Engine
speed
           else if(row == 9+m_vSpecificationData[6]) d_tmpSpecValue=DEF_MAX_OUTPUT_RATIO;   //Max output
ratio
        }

        m_vSpecificationData.push_back(d_tmpSpecValue);
    }
    fclose(fp_Specfile);

    //--------------------------------
    // If no gear has been detected
    //--------------------------------
    if(m_vSpecificationData[6]<=0) return ERROR_SPEC_DATA_FORMAT;

    //--------------------------------
    // If no data could be recorded
    //--------------------------------
    if(m_vSpecificationData.empty()) return ERROR_SPEC_FILE_EMPTY;

    return OK;
}
/**/
/*****************************************************************
 * Function name           : SaveTorqueData
 * Function summary        : Copies torque values in locale vector
 * Explanation             :
 *                         :
 * Argument (input)        : None
 * Argument (output)       : None
 * Argument (I/O)          : None
 * Return value            : 1: success ; others:failure, an error code is returned
 * Created by              :
 * Updated on (created on) :
 * Remarks                 :
 *****************************************************************/
int TCalculateProc::SaveTorqueData(string fileName)
{
    FILE *fp_torquefile;
    stTorqueData tmp_torqueData;
    char *p;
    char tmp_lineRead[LINE_MAX_LENGTH];
    memset(tmp_lineRead, 0x00, LINE_MAX_LENGTH ); //Reset of tmp_lineRead

    //--------------------------------
    //Opens and Reads the Envfile
    //--------------------------------
    fp_torquefile = fopen(fileName.c_str() , "r" );
    if((fp_torquefile == NULL)||(ferror(fp_torquefile)))
        return ERROR_TORQUE_FILE_NOT_FOUND;

    //--------------------------------
    //Reading the file and storing elements in structure
    //--------------------------------
    for(int row=0; fgets(tmp_lineRead, LINE_MAX_LENGTH, fp_torquefile); row++){
        if(((int)tmp_lineRead[0] < 0x30) || ((int)tmp_lineRead[0])> 0x39) continue;//If not number, line is
not read

        //Rotation Number
        p = strtok(tmp_lineRead, " ¥t,;%¥n");
        if( p == NULL ) continue;//Incomplete or empty line is ignored
        tmp_torqueData.d_EngineRevolutions=atof(p);

        //Torque value
        p = strtok(NULL, "¥n¥t");
        if( p == NULL ) continue;//Incomplete or empty line is ignored
        tmp_torqueData.d_EngineTorque=atof(p);

        m_MaxTorque.insert( tmp_torqueData );
    }
    fclose(fp_torquefile);

    //--------------------------------
    // If no data has been recorded
    //--------------------------------
    if(m_MaxTorque.empty()) return ERROR_TORQUE_FILE_EMPTY;

    return OK;
}

/**/
/*****************************************************************
 * Function name           : GetBestEngineSpeedMaintainGear
 * Function summary        : Calculates the gear that can be maintained as long as possible
 * Explanation             : The best gear feets requirements bellow (by order of priority)
 *                         : 1-Guarantee engine rotation speed
 *                                                 : 2-Guarantee conservation of gear as long as
dGearHoldTime
 *                                                 : 3-Guarantee target speed as far as possible
 * Argument (input)        : p_first   : First pointer of analysed period
 * Argument (input)        : p_second  : last pointer of analysed period
 * Argument (input)        : iCurrentGear : first analysed gear
 * Argument (input)        : dGearHoldTime: required gear hold time
 * Argument (output)       : None
 * Argument (I/O)          : None
 * Return value            : An array of DEF_MAXGEAR double elements containing the time optained for
optimal gears
 * Created by              :
 * Updated on (created on) :
 * Remarks                 :
 ****************************************************************/
stOptimalGears TCalculateProc::GetBestEngineSpeedMaintainGear(  map<double,stCalculateData>::iterator
p_start,

map<double,stCalculateData>::iterator p_end,
                                                                        int iAskedGear,
                                                                        double dAskedGearHoldTime,
                                    double dHoldTimes)
{
    stOptimalGears result;
    memset( &result, 0x00, sizeof( result ));

        if(iAskedGear<1 || iAskedGear>m_nMaxGear)       return result;

    map<double,stCalculateData>::iterator p_previous;          // Previous data
    int tmpGear = iAskedGear;
    int tmpPrevGear;
    int tmpSettingGear;
    int i;
    int iFoundTimes;
        double dMaxHoldTime=-1;
        double dDifferenceSpeed=9999;
    double holdTimeTab[DEF_MAXGEAR][GEAR_HOLD_TIME];
    double diffrenceSpeedTab[DEF_MAXGEAR][GEAR_HOLD_TIME];
    bool targetSpeedFollow[DEF_MAXGEAR][GEAR_HOLD_TIME];
    bool gearPatternFollow[DEF_MAXGEAR][GEAR_HOLD_TIME];
    bool gearChangeNeed[DEF_MAXGEAR][GEAR_HOLD_TIME];

    memset( holdTimeTab, 0x00, sizeof( holdTimeTab ));
    memset( diffrenceSpeedTab, 0x00, sizeof(diffrenceSpeedTab) );
        memset(targetSpeedFollow, 0, DEF_MAXGEAR*sizeof(bool));
    memset(gearPatternFollow, 0, DEF_MAXGEAR*sizeof(bool));
    memset(gearChangeNeed, 0, DEF_MAXGEAR*sizeof(bool));

    p_previous = p_start;
    p_previous--;
    tmpPrevGear      = p_previous->second.nCalcGear;
    if( tmpPrevGear == 0 ){
        tmpPrevGear = iAskedGear;
    }
    //----------------------------------------------
    // Calculate gear maintain time for all gears above (and including) iAskedGear
    //----------------------------------------------
    iFoundTimes = -1;
    if( dHoldTimes == 0 ){
        iFoundTimes = 0;
        while(tmpGear<=m_nMaxGear)
        {
                holdTimeTab[tmpGear-1][0] = GetGearMaintainTime(p_start, p_end, tmpGear,
dAskedGearHoldTime,
                                    targetSpeedFollow[tmpGear-1][0],
                                    gearPatternFollow[tmpGear-1][0],
                                    gearChangeNeed[tmpGear-1][0],
                                    diffrenceSpeedTab[tmpGear-1][0]);

                dMaxHoldTime = max( holdTimeTab[tmpGear-1][0], dMaxHoldTime);
            dDifferenceSpeed = min( diffrenceSpeedTab[tmpGear-1][0], dDifferenceSpeed );

                tmpGear++;
        }
    }else{
        tmpGear = iAskedGear;
        while(tmpGear<=m_nMaxGear)
        {
            for( i = 1; i < dHoldTimes+1; i++ ){
                    holdTimeTab[tmpGear-1][i-1] = GetGearMaintainTime(p_start, p_end, tmpGear,
dAskedGearHoldTime,
                                                        targetSpeedFollow[tmpGear-1][i-1],
                                                        gearPatternFollow[tmpGear-1][i-1],
                                                        gearChangeNeed[tmpGear-1][i-1],
                                                        diffrenceSpeedTab[tmpGear-1][i-1], i);

                dMaxHoldTime = max( holdTimeTab[tmpGear-1][i-1], dMaxHoldTime);
                if( ( gearChangeNeed[tmpGear-1][0] == false )&&
                    ( gearChangeNeed[tmpGear-1][i-1] == false )){
                    dDifferenceSpeed = min( diffrenceSpeedTab[tmpGear-1][i-1], dDifferenceSpeed );
                }
            }
            tmpGear++;
        }

        tmpGear = iAskedGear;
        iFoundTimes = -1;
        while(tmpGear<=m_nMaxGear)
        {
            for( i = 0; i < dHoldTimes; i++ ){
                if(( tmpGear == tmpPrevGear )&&
                    (gearChangeNeed[tmpPrevGear-1][0] == true )){
                    break;
                }
                if(( dDifferenceSpeed ==  diffrenceSpeedTab[tmpGear-1][i] )&&
                    ( gearChangeNeed[tmpGear-1][0] == false )&&
                    ( gearChangeNeed[tmpGear-1][i] == false )&&
                    ( holdTimeTab[tmpGear-1][0] != 0 )&&
                    ( holdTimeTab[tmpGear-1][i] == dHoldTimes )&&
                    ( holdTimeTab[tmpGear-1][i] != 0 )){
                    dMaxHoldTime = holdTimeTab[tmpGear-1][i];
                    iFoundTimes = i;
                    break;
                }
            }
            if( iFoundTimes != -1 ){
                break;
            }
            tmpGear++;
        }

        // Max hold time is best gear
        if( iFoundTimes == -1 ){
            tmpGear = iAskedGear;
            dMaxHoldTime = 0;
            tmpSettingGear = -1;
            dDifferenceSpeed=9999;
            while(tmpGear<=m_nMaxGear)
            {
                for( i = 0; i < dHoldTimes; i++ ){
                    if( ( holdTimeTab[tmpGear-1][i] == dAskedGearHoldTime )&&
                        ( dDifferenceSpeed > min( diffrenceSpeedTab[tmpGear-1][i], dDifferenceSpeed ) )&&
                        ( gearChangeNeed[tmpGear-1][0] == false )&&
                        ( gearChangeNeed[tmpGear-1][i] == false )&&
                        ( holdTimeTab[tmpGear-1][0] != 0 )&&
                        ( holdTimeTab[tmpGear-1][i] != 0 )){
                            dMaxHoldTime = holdTimeTab[tmpGear-1][i];
                            iFoundTimes = i;
                            dDifferenceSpeed = min( diffrenceSpeedTab[tmpGear-1][i], dDifferenceSpeed );
                            tmpSettingGear = tmpGear;
                    }
                }
                tmpGear++;
            }
            if( tmpSettingGear != -1 ){
                tmpGear = tmpSettingGear;
            }
        }

        // diffrenet speed min. is better.
        if( iFoundTimes == -1 ){
            tmpGear = iAskedGear;
            dMaxHoldTime = 0;
            tmpSettingGear = -1;
            dDifferenceSpeed=9999;
            while(tmpGear<=m_nMaxGear)
```

```
            {
                for( i = 0; i < dHoldTimes; i++ ){
                    if( ( dMaxHoldTime < max(holdTimeTab[tmpGear-1][i], dMaxHoldTime ) )&&
                        ( dDifferenceSpeed > min( diffrenceSpeedTab[tmpGear-1][i], dDifferenceSpeed ) )&&
                        ( gearChangeNeed[tmpGear-1][0] == false )&&
                        ( gearChangeNeed[tmpGear-1][i] == false )&&
                        ( holdTimeTab[tmpGear-1][0] != 0 )&&
                        ( holdTimeTab[tmpGear-1][i] != 0 )){
                            dMaxHoldTime = holdTimeTab[tmpGear-1][i];
                            iFoundTimes = i;
                            dDifferenceSpeed = min( diffrenceSpeedTab[tmpGear-1][i], dDifferenceSpeed );
                            tmpSettingGear = tmpGear;
                    }
                }
                tmpGear++;
            }
            if( tmpSettingGear != -1 ){
                tmpGear = tmpSettingGear;
            }
        }

        if( iFoundTimes == -1 ){
            tmpGear = iAskedGear;
            dMaxHoldTime = 0;
            tmpSettingGear = -1;
            dDifferenceSpeed=9999;
            while(tmpGear<=m_nMaxGear)
            {
                for( i = 0; i < dHoldTimes; i++ ){
                    if( ( dMaxHoldTime < max(holdTimeTab[tmpGear-1][i], dMaxHoldTime ) )&&
                        ( gearChangeNeed[tmpGear-1][0] == false )&&
                        ( gearChangeNeed[tmpGear-1][i] == false )&&
                        ( holdTimeTab[tmpGear-1][0] != 0 )&&
                        ( holdTimeTab[tmpGear-1][i] != 0 )){
                            dMaxHoldTime = holdTimeTab[tmpGear-1][i];
                            iFoundTimes = i;
                            dDifferenceSpeed = min( diffrenceSpeedTab[tmpGear-1][i], dDifferenceSpeed );
                            tmpSettingGear = tmpGear;
                    }
                }
                tmpGear++;
            }
            if( tmpSettingGear != -1 ){
                tmpGear = tmpSettingGear;
            }
        }

        if( iFoundTimes == -1 ){
            iFoundTimes = 0;
            dHoldTimes = 0;
            dMaxHoldTime = -1;
            dDifferenceSpeed = 9999;
            tmpGear = iAskedGear;
            while(tmpGear<=m_nMaxGear)
            {
                if(( tmpGear == tmpPrevGear )&&
                    (gearChangeNeed[tmpPrevGear-1][0] == true )){
                    tmpGear++;
                    continue;
                }
                dMaxHoldTime = max( holdTimeTab[tmpGear-1][i], dMaxHoldTime);
                if( ( gearChangeNeed[tmpGear-1][0] == false )&&
                    ( gearChangeNeed[tmpGear-1][i] == false )){
                    dDifferenceSpeed = min( diffrenceSpeedTab[tmpGear-1][i], dDifferenceSpeed );
                }
                    tmpGear++;
            }
        }
    }

    if( dMaxHoldTime != 0 ){
        if(( gearChangeNeed[tmpGear-1][iFoundTimes] == false )&&( dHoldTimes != 0 )){
                if( iFoundTimes != 0 ){
                    if((gearChangeNeed[tmpPrevGear-1][0] == true )){
                        result.iGearsID[result.iGearsNb] = tmpGear;
                    }else{
                        result.iGearsID[result.iGearsNb] = tmpPrevGear;
                    }
                }else{
                    result.iGearsID[result.iGearsNb] = tmpGear;
                }
                result.dMaintainTime[result.iGearsNb] = holdTimeTab[tmpGear-1][iFoundTimes];
                result.bTargetSpeedFollowed[result.iGearsNb] = targetSpeedFollow[tmpGear-1][iFoundTimes];
                result.bGearPatternFollowed[result.iGearsNb] = gearPatternFollow[tmpGear-1][iFoundTimes];
                result.bGearChangeNeeded[result.iGearsNb] = gearChangeNeed[tmpGear-1][iFoundTimes];
                result.bBestMaintainTime[result.iGearsNb] = true;
                result.iGearsNb++;
        }else if( gearChangeNeed[iAskedGear-1][iFoundTimes] == true ){
            result.iGearsNb = 0;
            for( i = iAskedGear; i < DEF_MAXGEAR; i++ ){
                if((holdTimeTab[i-1][iFoundTimes]==dAskedGearHoldTime)&&
                            (targetSpeedFollow[i-1][iFoundTimes] == true )&&
                            ( gearChangeNeed[i-1][iFoundTimes] == false )){
                    result.iGearsID[result.iGearsNb] = i;
                    result.dMaintainTime[result.iGearsNb] = holdTimeTab[i-1][iFoundTimes];
                    result.bTargetSpeedFollowed[result.iGearsNb] = targetSpeedFollow[i-1][iFoundTimes];
                    result.bGearPatternFollowed[result.iGearsNb] = gearPatternFollow[i-1][iFoundTimes];
                    result.bGearChangeNeeded[result.iGearsNb] = gearChangeNeed[i];
                    result.bBestMaintainTime[result.iGearsNb] = true;
                    result.iGearsNb++;
                }
            }
            if( result.iGearsNb == 0 ){
                for( i = iAskedGear; i < DEF_MAXGEAR; i++ ){
                    if((holdTimeTab[i-1][iFoundTimes]==dAskedGearHoldTime)&&
                                        ( gearChangeNeed[i-1][iFoundTimes] == false )){
                        if( diffrenceSpeedTab[i-1][iFoundTimes] == dDifferenceSpeed ){
                            result.iGearsID[result.iGearsNb] = i;
                            result.dMaintainTime[result.iGearsNb] = holdTimeTab[i-1][iFoundTimes];
                            result.bTargetSpeedFollowed[result.iGearsNb] =
targetSpeedFollow[i-1][iFoundTimes];
                            result.bGearPatternFollowed[result.iGearsNb] =
gearPatternFollow[i-1][iFoundTimes];
                            result.bGearChangeNeeded[result.iGearsNb] = gearChangeNeed[i-1][iFoundTimes];
                            result.bBestMaintainTime[result.iGearsNb] = true;
                            result.iGearsNb++;
                        }
                    }
                }
            }
            if( result.iGearsNb == 0 ){
                for( i = iAskedGear; i < DEF_MAXGEAR; i++ ){
                    if((holdTimeTab[i-1][iFoundTimes]==dMaxHoldTime)&&
                                        ( gearChangeNeed[i-1][iFoundTimes] == false )){
                        result.iGearsID[result.iGearsNb] = i;
                        result.dMaintainTime[result.iGearsNb] = holdTimeTab[i-1][iFoundTimes];
                        result.bTargetSpeedFollowed[result.iGearsNb] = targetSpeedFollow[i-1][iFoundTimes];
                        result.bGearPatternFollowed[result.iGearsNb] = gearPatternFollow[i-1][iFoundTimes];
                        result.bGearChangeNeeded[result.iGearsNb] = gearChangeNeed[i-1][iFoundTimes];
                        result.bBestMaintainTime[result.iGearsNb] = true;
                        result.iGearsNb++;
                    }
                }
            }
        }else if( gearPatternFollow[iAskedGear-1][iFoundTimes] == false ){
            result.iGearsNb = 0;
            for( i = iAskedGear; i < DEF_MAXGEAR; i++ ){
                if((holdTimeTab[i-1][iFoundTimes]==dAskedGearHoldTime)&&
                            ( gearPatternFollow[i-1][iFoundTimes] == true )&&
                            (targetSpeedFollow[i-1][iFoundTimes] == true )&&
                            ( gearChangeNeed[i-1][iFoundTimes] == false )){
                    result.iGearsID[result.iGearsNb] = i;
                    result.dMaintainTime[result.iGearsNb] = holdTimeTab[i-1][iFoundTimes];
                    result.bTargetSpeedFollowed[result.iGearsNb] = targetSpeedFollow[i-1][iFoundTimes];
                    result.bGearPatternFollowed[result.iGearsNb] = gearPatternFollow[i-1][iFoundTimes];
                    result.bGearChangeNeeded[result.iGearsNb] = gearChangeNeed[i-1];
                    result.bBestMaintainTime[result.iGearsNb] = true;
                    result.iGearsNb++;
                }
            }
            if(( result.iGearsNb == 0 )&&(dHoldTimes != 0 )){
                for( i = iAskedGear; i < DEF_MAXGEAR; i++ ){
                    if((holdTimeTab[i-1][iFoundTimes]==dAskedGearHoldTime)&&
                            ( gearPatternFollow[i-1][iFoundTimes] == true )&&
                            ( gearChangeNeed[i-1][iFoundTimes] == false )){
                        if( diffrenceSpeedTab[i-1][iFoundTimes] == dDifferenceSpeed ){
                            result.iGearsID[result.iGearsNb] = i;
                            result.dMaintainTime[result.iGearsNb] = holdTimeTab[i-1][iFoundTimes];
                            result.bTargetSpeedFollowed[result.iGearsNb] =
targetSpeedFollow[i-1][iFoundTimes];
                            result.bGearPatternFollowed[result.iGearsNb] =
gearPatternFollow[i-1][iFoundTimes];
                            result.bGearChangeNeeded[result.iGearsNb] = gearChangeNeed[i-1][iFoundTimes];
                            result.bBestMaintainTime[result.iGearsNb] = true;
                            result.iGearsNb++;
                        }
                    }
                }
            }
        }
    }

    return result;
}

/**/
/*****************************************************************
 * Function name          : GetGearMaintainTime
 * Function summary       : Calculates the time asked gear can be maintained
 * Explanation            :
 * Argument (input)       : p_start   : First pointer of analysed period
 * Argument (input)       : p_end  : last pointer of analysed period
 * Argument (input)       : iGear : first analysed gear
 * Argument (input)       : dGearHoldTime: required gear hold time
 * Argument (input)       : bTargetSpeedFollowed
 * Argument (input)       : bGearPatternFollowed
 * Argument (input)       : bGearChangeNeed
 * Argument (input)       : dDifferenceSpeed
 * Argument (input)       : iShiftChangeTimes
 * Argument (output)      : None
 * Argument (I/O)         : None
 * Return value           : An array of DEF_MAXGEAR double elements containing the time optained for
optimal gears
 * Created by             :
 * Updated on (created on) :
 * Remarks                :
 ****************************************************************/
double TCalculateProc::GetGearMaintainTime(     map<double,stCalculateData>::iterator p_start,

map<double,stCalculateData>::iterator p_end,
                                                                                         int iGear,
                                                                                         double
dGearHoldTime,
                                            bool &bTargetSpeedFollowed,
                                            bool &bGearPatternFollowed,
                                            bool &bGearChangeNeed,
                                            double &dDifferenceSpeed,
                                            int iShiftChangeTimes)
{
        map<double,stCalculateData>::iterator p_tmpCalculate;      // Temporary pointer
        double tmpNe;
    double dMaintainTime = 0;
    int    dHoldTimes;

    map<double,stCalculateData>::iterator p_previous;          // Previous data
    double fCarAcc, tmpPrevGearTime, tmpPrevCalcTime, tmpTe, tmpTeMax, tmpTargetSpeed, tmpPrevVAna_sp ;
    int tmpPrevGear, tmpNowGear = iGear;

    p_previous = p_start;
    p_previous--;
    tmpPrevVAna_sp = p_previous->second.fVAna_sp;         // Previous analysis speed
    tmpPrevGearTime = p_previous->second.nGearTime;        // Previous gear time
    tmpPrevGear     = p_previous->second.nCalcGear;
    if( tmpPrevGear == 0 ){
        tmpPrevGear = iGear;
    }


    bTargetSpeedFollowed = true;
    bGearPatternFollowed = true;
    bGearChangeNeed = false;
    dDifferenceSpeed = 0;

    dHoldTimes = 0;

    for(p_tmpCalculate=p_start ; p_tmpCalculate!=p_end ; p_tmpCalculate++)
    {
        tmpTargetSpeed  = p_tmpCalculate->second.fVTarget_sp;

        p_previous = p_tmpCalculate;
        p_previous--;
        tmpPrevCalcTime = p_previous->second.nCalcTime;        // Previous required time

        //--------------------------------
        // Verify gear change rules
        //--------------------------------
        if( ((iShiftChangeTimes == 0 )&&( p_tmpCalculate == p_start )) ){
            //--------------------------------
            // Test if gear change has not been done too recently (shortest period, in sec, is 
GEAR_HOLD_TIME)
            //--------------------------------
            if(((tmpPrevGearTime + tmpPrevCalcTime)/10.0)>GEAR_HOLD_TIME){

                //--------------------------------
                // In case of deceleration previously
                //--------------------------------
                if(p_previous->second.nFlag == ENGINE_DECELERATE){
                    if( (tmpTargetSpeed<10.0) && (tmpNowGear>1+m_nPtnGearUp) ){
                        if(1+m_nPtnGearUp<=m_nMaxGear){
                            if( tmpPrevGear < 1 + m_nPtnGearUp ){
                                tmpNowGear = 1 + m_nPtnGearUp;
                                bGearPatternFollowed = false;
                            }
                        }
                    }
                    else if( (tmpTargetSpeed<20.0) && (tmpNowGear>2+m_nPtnGearUp) ){
                        if(2+m_nPtnGearUp<=m_nMaxGear){
                            if( tmpPrevGear < 2 + m_nPtnGearUp ){
                                tmpNowGear = 2 + m_nPtnGearUp;
                                bGearPatternFollowed = false;
                            }
                        }
                    }
                    else 
if((tmpTargetSpeed<40.0)&&(tmpNowGear>3+m_nPtnGearUp)&&(3+m_nPtnGearUp<=m_nMaxGear)){
                        if( tmpPrevGear < 3 + m_nPtnGearUp ){
                            tmpNowGear = 3 + m_nPtnGearUp;
                            bGearPatternFollowed = false;
                        }
                    }
                    else 
if((tmpTargetSpeed<60.0)&&(tmpNowGear>4+m_nPtnGearUp)&&(4+m_nPtnGearUp<=m_nMaxGear)){
                        if( tmpPrevGear < 4 + m_nPtnGearUp ){
                            tmpNowGear = 4 + m_nPtnGearUp;
                            bGearPatternFollowed = false;
                        }
                    }
                }
                //--------------------------------
                // In case of acceleration previously
                //--------------------------------
                else{

if((tmpTargetSpeed>15.0)&&(tmpNowGear<2+m_nPtnGearUp)&&(2+m_nPtnGearUp<=m_nMaxGear)&&(2+m_nPtnGearUp>tmpPre
vGear)){
                        tmpNowGear = 2 + m_nPtnGearUp;
                        bGearPatternFollowed = false;
                    }
                    else 
if((tmpTargetSpeed>30.0)&&(tmpNowGear<3+m_nPtnGearUp)&&(3+m_nPtnGearUp<=m_nMaxGear)&&(3+m_nPtnGearUp>tmpPre
vGear)){
                        tmpNowGear = 3 + m_nPtnGearUp;
                        bGearPatternFollowed = false;
                    }
                    else
if((tmpTargetSpeed>50.0)&&(tmpNowGear<4+m_nPtnGearUp)&&(4+m_nPtnGearUp<=m_nMaxGear)&&(4+m_nPtnGearUp>tmpPre
vGear)){
                        tmpNowGear = 4 + m_nPtnGearUp;
                        bGearPatternFollowed = false;
                    }
                    else
if((tmpTargetSpeed>70.0)&&(tmpNowGear<5+m_nPtnGearUp)&&(5+m_nPtnGearUp<=m_nMaxGear)&&(5+m_nPtnGearUp>tmpPre
vGear)){
                        tmpNowGear = 5 + m_nPtnGearUp;
                        bGearPatternFollowed = false;
                    }
                }
            }
        }

                //--------------------------------
                // With current gear, calculate engine speed
                //--------------------------------
        if((iShiftChangeTimes == 0 )&&( p_tmpCalculate == p_start )){
            GetNe(tmpNowGear, p_tmpCalculate->second.fVTarget_sp ,tmpNe);
        }else if((iShiftChangeTimes != 0 )&&( dHoldTimes < iShiftChangeTimes-1 )){
            GetNe(tmpPrevGear, p_tmpCalculate->second.fVTarget_sp ,tmpNe);
        }else{
            GetNe(tmpNowGear, p_tmpCalculate->second.fVTarget_sp ,tmpNe);
        }

                if( tmpNe > m_fMaxOutputRotation){ // If maximum is reached: stop search
            if( iShiftChangeTimes == 0 ){
                if(tmpPrevGear != tmpNowGear ){
                    bGearChangeNeed = true;
                }
                if(( p_tmpCalculate == p_start )&&(tmpPrevGear == tmpNowGear )){
                    bGearChangeNeed = true;
                }
                break;
            }
        }

        // If engine speed is too low
        if(tmpNe<m_fClutch_MeetNe && (p_previous->second.fVAna_sp==0 ||
p_previous->second.bClutchMeetMode==true)){
            tmpNe=m_fClutch_MeetNe;
        }

        fCarAcc = ((tmpTargetSpeed - tmpPrevVAna_sp)/(tmpPrevCalcTime/10.0))/3.6;
        if((iShiftChangeTimes == 0 )&&( p_tmpCalculate == p_start )){
            GetNe(tmpNowGear, p_tmpCalculate->second.fVTarget_sp ,tmpNe);
            GetTe(tmpNowGear,tmpTargetSpeed,fCarAcc, tmpTe);
        }else if((iShiftChangeTimes != 0 )&&( dHoldTimes < iShiftChangeTimes-1 )){
            GetTe(tmpPrevGear,tmpTargetSpeed,fCarAcc, tmpTe);
        }else{
            GetTe(tmpNowGear,tmpTargetSpeed,fCarAcc, tmpTe);
        }
        tmpTeMax = GetLineReviseMaxTorque(tmpNe);

        // Max trq check
        if(tmpTe>tmpTeMax){
            if((iShiftChangeTimes == 0 )&&( p_tmpCalculate == p_start )){
                CalcTeMaxSp(tmpNowGear, tmpPrevCalcTime, tmpPrevVAna_sp, tmpTargetSpeed, tmpNe, tmpTe );
            }else if((iShiftChangeTimes != 0 )&&( dHoldTimes < iShiftChangeTimes-1 )){
                CalcTeMaxSp(tmpPrevGear, tmpPrevCalcTime, tmpPrevVAna_sp, tmpTargetSpeed, tmpNe, tmpTe );
            }else{
                CalcTeMaxSp(tmpNowGear, tmpPrevCalcTime, tmpPrevVAna_sp, tmpTargetSpeed, tmpNe, tmpTe );
            }

            if(tmpPrevGear != tmpNowGear ){
                bTargetSpeedFollowed=false;
            }
            if(( p_tmpCalculate == p_start )&&(tmpPrevGear == tmpNowGear )){
                bTargetSpeedFollowed=false;
            }
            if( iShiftChangeTimes != 0 ){
                bTargetSpeedFollowed=false;
            }
        }
        // Max Ne check
                if( tmpNe >= m_fMaxOutputRotation){ // If maximum is reached: stop search
            if( iShiftChangeTimes == 0 ){
                if(tmpPrevGear != tmpNowGear ){
                    bGearChangeNeed = true;
                }
                if(( p_tmpCalculate == p_start )&&(tmpPrevGear == tmpNowGear )){
                    bGearChangeNeed = true;
                }
                break;
            }else{
                if( p_tmpCalculate == p_start ){
                    bGearChangeNeed = true;
                }
                if(tmpPrevGear > tmpNowGear ){
                    bGearChangeNeed = true;
                    break;
                }
                break;
            }
        }

        dDifferenceSpeed = dDifferenceSpeed + fabs(tmpTargetSpeed - p_tmpCalculate->second.fVTarget_sp );
        tmpPrevVAna_sp = tmpTargetSpeed;
        tmpPrevGearTime = tmpPrevGearTime + p_tmpCalculate->second.nCalcTime;


        //Update maintain time
        dMaintainTime += p_tmpCalculate->second.nCalcTime/10;
        if( dMaintainTime>=dGearHoldTime ){
            break;
        }
        dHoldTimes++;
        }

    bGearPatternFollowed = bGearPatternFollowed && dMaintainTime!=0;

    return dMaintainTime;
}
/**/
/*****************************************************************
 * Function name          : main
 * Function summary       : Main processing
 * Explanation            : Main process of conversion processing
 *                        :
 * Argument (input)       : None
 * Argument (output)      : None
 * Argument (I/O)         : None
 * Return value           : None
 * Created by             :
 * Updated on (created on) :
 * Remarks                :
 *****************************************************************/
#ifndef __GNUC__
int __cdecl main(int argc, char* argv[])
#else
int main(int argc, char* argv[])
#endif
{
    int  nRet;
    bool bRet;
    string runningMode;

    CalculateProc = new TCalculateProc(); // Initialization

    //---------------------------------
    // Verifying if arguments are present or not
    //---------------------------------
    if( argc == 2 ){//Only the output file is known
        CalculateProc->setInputFileName(DEF_MAIN_ENVFILE);
        CalculateProc->setOutputFileName(string(argv[1]));
    }
    else if( argc >= 3 ){//both input and output file are known
        CalculateProc->setInputFileName(string(argv[1]));
        CalculateProc->setOutputFileName(string(argv[2]));
    }
    else{//Neither input file nor output file
        CalculateProc->setInputFileName(string(DEF_MAIN_ENVFILE));
        CalculateProc->setOutputFileName(string(""));
    }

    //Record of input data contained in files listed in Main_Envfile
    nRet=CalculateProc->Data_Acquisition();

    //--------------------------------
    //In case of error a specific message is displayed
    //--------------------------------
    if(nRet!=OK) {
        cout << "Error encountered !" << endl;
        if(nRet==ERROR_MAIN_FILE_NOT_FOUND) cout << ERROR_MAIN_FILE_NOT_FOUND_STR
<<"("<<CalculateProc->getInputFileName()<<")"<< endl;
        else if(nRet==ERROR_ENV_FILE_NOT_FOUND) cout << ERROR_ENV_FILE_NOT_FOUND_STR << endl;
        else if(nRet==ERROR_SPEC_FILE_NOT_FOUND) cout << ERROR_SPEC_FILE_NOT_FOUND_STR << endl;
        else if(nRet==ERROR_TORQUE_FILE_NOT_FOUND) cout << ERROR_TORQUE_FILE_NOT_FOUND_STR << endl;
        else if(nRet==ERROR_ENV_FILE_EMPTY) cout << ERROR_ENV_FILE_EMPTY_STR << endl;
        else if(nRet==ERROR_SPEC_FILE_EMPTY) cout << ERROR_SPEC_FILE_EMPTY_STR << endl;
        else if(nRet==ERROR_SPEC_FILE_EMPTY) cout << ERROR_SPEC_FILE_EMPTY_STR << endl;
        else if(nRet==ERROR_SPEC_DATA_FORMAT) cout << ERROR_SPEC_DATA_FORMAT_STR << endl;

        exit(-1);
    }

    //--------------------------------
    //Initialize some specification datas
    //--------------------------------
    bRet = CalculateProc->Init();
    if( bRet == false ){
        cout << "Stopped with error." << endl;
        exit(-1);
    }

    //--------------------------------
    // Conversion infomation
    //--------------------------------
    cout << "Ver " << MY_VERSION << endl;
    cout << "Convert start!" << endl;
    nRet = CalculateProc->CalculateProcess();   // Initiates conversion processing.
    if( nRet == NG ){
            cout << "Stopped with error during calculation process." << endl;
            exit(-1);
    }
    cout << "Convertion finished!" << endl;

    // Post-processing
    delete CalculateProc;

    exit(0);
    return(0);
}
#endif
```