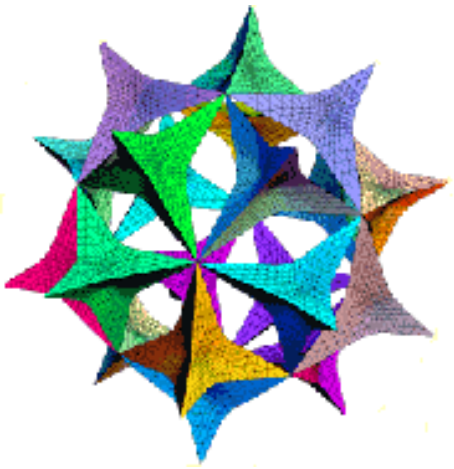


Easy *Math* Solution

EquTranslator (Version 2.0)

Context

Introduction

Thank you for your interest in ***Easy Math Solution*** products. *EquTranslator* is a fast equation parser-calculator with parse-tree builder and user-friendly interface for parsing and calculation a run-time defined math expression.
The math expressions is represented as a string in the function style

F(x1, x2, …, xn) = f(x1, x2, …, xn)

, where

F(x1, x2, …, xn) - declares number and names of the variables;

f(x1, x2, …, xn) – math expression which may contain:

- variables x1, x2, …, xn;
- arithmetic operations: + - * / ^;
- logical operators: <, >, =, # (not equal), &(logical AND), | (logical OR);
- functions*: abs(x), acos(x), asin(x), atan(x), atan2(y,x), ceil(x), cos(x), cosh(x), exp(x), floor(x), fmod(x,y), J0(x), J1(x), JN(n,x), hypot(x,y), iif(cond, exp1, exp2), ln(x), log10(x), max(x,y), min(x,y), sin(x), sinh(x), sqrt(x), tan(x), tanh(x), Y0(x), Y1(x), YN(n,x);*
- constants *e* = 2.71828182845904 and *pi* = 3.14159265358979, and ;
- user defined functions and constants.

For example:

*F(x, y) = 3*x^2+y^2+e/4,*
F(x1, x2, y) = (x1-x2)/iif(x1>x2+y,sin(x1), cos(y))
F()=5+6+pi.

The algorithm has been developed and implemented by ***Easy Math Solution*** and has different binary (static link library, dynamic link library and COM) implementation and programming approach (procedural and object oriented). It allows using *EquTranslator* in C/C++, Delphi, VB, .NET and other languages and systems that can understand dll's calls and COM.

The main characteristics of *EquTranslator*:

- extremely **fast**,
- procedural and OOP implementation,
- multithreading (OOP version only),
- unlimited number of variables,
- exception mechanism provided,
- ASCII and UNICODE supports,
- can be extended by user defined functions and constants.

EquTranslator Functions

Function	Use
abs(x)	Find absolute value
acos(x)	Calculate arccosine
asin(x)	Calculate arcsine
atan(x)	Calculate arctangent
atan2(y,x)	Calculate arctangent of y/x
ceil(x)	Returns the smallest value that is greater than or equal to *x: **ceil**(3.5) =4*
cos(x)	Calculate cosine
cosh(x)	Calculate hyperbolic cosine
exp(x)	Calculate exponential function
floor(x)	Find largest integer less than or equal to argument: ***floor***(3.5)=3
fmod(x,y)	Find floating-point remainder: ***fmod***(-10, 3)=-1
J0(x), J1(x), JN(n,x)	Return **Bessel** functions of the first kind: orders 0, 1, and *n*, respectively
hypot(x,y)	Calculate hypotenuse of right triangle. A call to **hypot** is equivalent to the square root of $x^2 + y^2$.
iif(cond, exp1, exp2)	Calculate *exp1*, if *cond* is true, else *exp2*.
ln(x)	Calculate natural logarithm
log10(x)	Calculate base-10 logarithm.
max(x,y)	Return larger of two values
min(x,y)	Return smaller of two values
sin(x)	Calculate sine
sinh(x)	Calculate hyperbolic sine
sqrt(x)	Find square root
tan(x)	Calculate tangent
tanh(x)	Calculate hyperbolic tangent
Y0(x), Y1(x), YN(n,x)	Return **Bessel** functions of the second kind: orders 0, 1, and *n*, respectively

Try the DEMO version of *EquTranslator* and check it out if your calculation algorithm needs some improvements.
Please, review our web site www.e-MathSolution.com regularly for new products and services.
For more detail information, please contact us at:
info@e-MathSolution.com

Price and licenses

As it was mentioned earlier, *EquTranslator* algorithm has different implementation. In this chapter we describe tags, which represent variety of the implementations, and type of the licenses.
So, *EquTranslator* is implemented as:

- Static Link Library (lib), both procedural and object oriented programming approach;
- Dynamic Link Library (dll), both procedural and object oriented programming approach.

Each type of the implementation is tagged on purpose to distinguish different products and to avoid mistakes in the Order Form.
A tag consists of 3 fields separated by point:

xxxxx.xxx.xxx

1st position is a name of the component. Here is EquTr.
2nd position is a binary implementation: lib or dll.
3rd position is a programming technique: PRC describes procedural, OOP – object oriented.

License options include Home, Academic, and Professional. Before selecting a license type, please read this part thoroughly.

Home License

With this type of License, *EquTranslator* can be used for personal projects only. It covers software development process with *EquTranslator* on personally owned computer(s) without any type of a commercial use and distribution.

Academic License

This License is for non-commercial use with in an educational institution and for educational process only.

Professional License

EquTranslator can be included royalty-free in commercially distributed projects as well as non-commercial.

Table below presents prices on different implementations of *EquTranslator* and licenses.

All prices are given in US dollars

Tag \ License	Home	Academic	Professional	Use
EquTr.lib.PRC				VC++
EquTr.lib.OOP				VC++
EquTr.dll.PRC				VC++, Delphi, VB, C++Builder, .NET…
EquTr.dll.OOP				VC++

*To get the right price for *EquTranslator*, please, visit: www.e-MathSolution.com.

Each purchase includes ASCII version as well as UNICODE version.
The UNICODE version of the algorithm has postfix "W":
EquTranslatorW.lib or EquTranslatorW.dll
Want to get more information, please send request to license@e-MathSolution.com .

Precedence and Order of Evaluation

When writing compound expressions, you should be explicit and indicate with parentheses which operators should be evaluated first. This pratice will make your code easier to read and to maintain. You can specify exactly how you want an expression to be evaluated, using balanced parentheses — **(** and **)**.
If you don't explicitly indicate the order in which you want the operations in a compound expression to be performed, the order is determined by the *precedence* assigned to the operators in use within the expression.
Operators with a higher precedence get evaluated first.
The following table shows the precedence assigned to the operators in *EquTranslator* algorithm. The operators in this table are listed in precedence order: The higher in the table an operator appears, the higher its precedence. Operators with higher precedence are evaluated before operators with a relatively lower precedence. Operators on the same line have equal precedence. When operators of equal precedence appear in the same expression, a rule must govern which is evaluated first.

Operator Precedence
^
*, /
+, -
<, >, =, #
&, |

Using the **iif** function

The **iif** function controls conditional parse/evaluation.
Syntax:

iif(*condition*, *expression1*, *expression2*);

condition – the expressions, which can be parsed and evaluated;
expression1 – the valid expression to be evaluated and returned if *condition* is TRUE;
expression2 – the valid expression to be evaluated and returned if *condition* is FALSE.

If *condition* is true (more then zero), *expression1*is evaluated. If *condition* is false (less or equal to zero), *expression2* is evaluated. It's mean that the **iif** function interprets first argument as regular value: **iif**(5, x, y) – returns x, but **iif**(0, x, y) – returns y. Also, user of EquTranslator should know that logical operators: <, >, =, #, &, and | - return 1 as TRUE, and –1 as FALSE.

Examples:
"F(x,y,z)=sin(x)+iif(x-5>y, y*z, x*z)";
"F(x,y,z)= iif(x-5>y | (z-x)/y<4, sin(iif(x#10, 5*x, cos(x))), x*z)";
"F(x,y,z)=exp(x)/iif(x-5, y*z, x*z)"

Interface

Both, procedural and object oriented versions have practically the same interface with a very little difference (see "Initialization and Error Handling").
First of all let's create some aliases, which will be used in description of the interface and it can help a reader better understand meaning of the each interface's function.
One of the important elements of the *EquTranslator* algorithm is a calculation type, which is defined as:

```
typedef double TCalc;
```

Type TCHAR is defined as a string element and can be either of the two regular character types char or wchar_t (Ansi or Unicode).
Also, let's assume that,
varList – a list of variables *x1, x2, …, xn* which must be used in a math expression, separated by either symbol from string ",:;" or white space. For example: "x, y, z", or "x;y;z", or "x y z";
mathFunc – a math expression in the function form *F(x1, x2, …, xn) = f(x1, x2, …, xn)* . For example:
"F(x1, x2, y) = (x1-x2)/y";
mathExp – a math expression *f(x1, x2, …, xn);*
args – an array of arguments of type **TCalc** to be calculated for *mathExp*.

TCalc EvaluateEqu(const TCHAR * mathFunc, const TCalc *args);

The function parses and evaluates a *mathFunc* expression for given values of an argument (*args*).
Example:

```
…
int main(int argc, char* argv[])
{
    TCalc args[] = {2, 5.6, -11};
    TCalc res;
    TCHAR *mathFunc = "F(x, y, z) = 3*x^2-y^2+z/4";
……..
    res = EvaluateEqu(mathFunc, args);
    cout<<"Res="<<res;
……
}
```

TCalc Evaluate(const TCHAR *varList, const TCHAR *mathExp, const TCalc *args);

The function parses and evaluates *mathExp* for given values of the argument (*args*). The argument *varList* consists of number and a list of variables for *mathExp*.
Example:

```
……..
int main(int argc, char* argv[])
{
      TCalc args[] = {2, 5.6, -11};
    TCalc res;
    TCHAR *mathExp = "3*x^2-y^2+z/4";
    TCHAR *varList = "x,y,z";
……..
    res = Evaluate(varList, mathExp, args);
    cout<<"Res="<<res;
……
}
```

```
void BuildEqu(const TCHAR * mathFunc);
void Build(const TCHAR *varList, const TCHAR *mathExp,);
TCalc CalcFor(const TCalc *args);
```

BuildEqu and **Build** are allowed to build an optimized image of the *mathExp* and get result with function **CalcFor** for given arguments.

```
…..
int main(int argc, char* argv[])
{
  TCalc args[] = {2, 5.6, -11};
  TCalc res;
 TCHAR *mathFunc = "F(x, y, z) = 3*x^2-y^2+z/4";
  TCHAR *mathExp = "3*x^2-y^2+z/4";
  TCHAR *varList = "x y z";
…….
    BuildEqu(mathFunc);
    res = CalcFor(args);
    cout<<"Res="<<res<<endl;

    Build(varList, mathExp);
    res = CalcFor(args);
    cout<<"Res="<<res<<endl;
…..
}
```

```
bool AddFunction(const TCHAR*funName, const TCalc (__stdcall *)(const TCalc &));
bool RemoveFunction(const TCHAR*funName);
```

To add and remove user defined *function*. The *function* must have __stdcall convention type. Returns true if success.
Note: The EquTranslator algorithm can be recompiled and distributed with cdecl user defined function if customers request it (charge free).

```
…
const TCalc __stdcall toRad(const TCalc &x)   //User defined function
{
          return x*3.14/180;
}

int main(int argc, char* argv[])
{
TCalc args[] = {2, 5.6, -11};
TCalc res;
TCHAR *mathFunc = "F(x, y, z) = 3*x^2-y^2+sin(toRad(45))/4";
TCHAR *mathExp = "3*x^2-y^2+cos(toRad(85))/4";
TCHAR *varList = "x y z";
……
AddFunction("toRad", toRad);
BuildEqu(mathFunc);
res = CalcFor(args);
cout<<"Res="<<res<<endl;

RemoveFunction("toRad");
Build(varList, mathExp);   //error "Uknown function: toRad" occurs
res = CalcFor(args);
cout<<"Res="<<res<<endl;
…..
}
```

```
bool AddConst(const TCHAR*constName, TCalc constVal);
bool RemoveConst(const TCHAR*constName);
```

To add and remove user defined *const*. Returns true if success.

```
…
int main(int argc, char* argv[])
{
      TCalc args[] = {2, 5.6, -11};
  TCalc res;
  TCalc two_pi = 2*3.14;
  TCHAR *mathFunc = "F(x, y, z) = 3*x^2-y^2+z/twoPi";
  TCHAR *mathExp = "3*x^2-y^2+z/twoPi";
  TCHAR *varList = "x y z";
…..
   AddConst("twoPi", two_pi);
   BuildEqu(mathFunc);
   res = CalcFor(args);
   cout<<"Res="<<res<<endl;

   RemoveConst("twoPi");
   Build(varList, mathExp); //error "Uknown function: twoPi" occurs
   res = CalcFor(args);
   cout<<"Res="<<res<<endl;
…
}
```

As you can see, the interface is easy to use and human friendly.

Initialization and Error Handling

Initialization

Initialization is a very important phase of *EquTranslator* use. Some times initialization happens without programmer's awareness, but in most cases programmer should do some steps first to use *EquTranslator* for parse and calculations.
In general *EquTranslator* has two functions
void InitEquTranslator(void) and void CloseEquTranslator(void),
and, it depends on the implementation and programming approach, both, one or none functions may be called.
So, in case of:

- EquTr.lib.PRC - both function must be called: InitEquTranslator() and CloseEquTranslator();
- EquTr.lib.OOP – programmer must call InitEquTranslator() only;
- EquTr.dll.PRC – the implicit and explicit call of EquTranslator.dll involve InitEquTranslator() and CloseEquTranslator() automatically. Some programming languages and systems require including in project distributed with EquTranslator source modules that have

subroutines to load dll and to initialize interface (see examples);
- EquTr.dll.OOP – to call InitEquTranslator() and CloseEquTranslator() is not required.

Error Handling

Some languages provide built-in support for handling anomalous situations, known as "exceptions," which may occur during the execution of a program. With exception handling, the program can communicate unexpected events to a higher execution context that is better able to recover from such abnormal events. These exceptions are handled by code that is outside the normal flow of control.
The problems raise when we use exception mechanism within dll and try catches error outside of the dll that is very important in parsing and calculating the math expression. Especially if an application which is using dll was developed with another language or within another environment.
EquTranslator uses exception mechanism to locate error and pass message of the error that occurs.
But for Delphi, VB, C#, VB.NET or even Borland C++ application to get an error message through exception mechanism from a VC++ dynamic link library is obviously a challenge.
There was developed mechanism to catch error message using a **Callback** function and raise native exception inside calling program.
For that purposes in the interface has been included function:
void SetErrorHandle(TpfErrorHandler);
with argument
typedef void (__stdcall*TpfErrorHandler)(const TCHAR*errMsg);
pointer to the function with one argument const TCHAR *, a string with error message.
In case if programmer wants to provide his own exception mechanism, he must implement a function that throws exception with error message from *EquTranslator*, for example:

```
void __stdcall RiaseException (const TCHAR*err)
{
        throw Exception(err);
}
```

Calling SetErrorHandle(RiaseException), programmer establishes mechanism to redirect all error messages from *EquTranslator* to void RiaseException (const TCHAR*err) function and as result, all errors can be easy caught in

```
Try{
...}
catch(Exception &e)
{...}
```

block inside of the calling function.
To make this clear, let's illustrate, how different languages and systems provide the initialization and the error handling using EquTr.dll.PRC version of *EquTranslator* algorithm.

Example 1: C/C++

For explicit use of EquTranslator(W).dll, head file EquTranslatorEx.h must be included in a project:

```
//EquTranslatorEx.h
//Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
//http://www.e-MathSolution.com
//For details contact us: info@e-MathSolution.com
#ifndef EQUTRANSLATOREX_H
#define EQUTRANSLATOREX_H

#include<TCHAR.H>
#include<windows.h>
#include<string>


//The type definitions
typedef double TCalc;
//CallBack function to handle the errors
typedef void (__stdcall*TpfErrorHandler)(const TCHAR*);
//Math functions. User defined functions
 typedef const TCalc (__stdcall *MATHFUNCTION_PTR)(const TCalc &);


typedef TCalc (__stdcall* TEvaluateEqu)(const TCHAR *exp, const TCalc *args);
typedef TCalc (__stdcall* TEvaluate)(const TCHAR *arg, const TCHAR *exp,
                        const TCalc *args );
typedef void (__stdcall* TBuildEqu)(const TCHAR *exp);
typedef void (__stdcall* TBuild)(const TCHAR *arg, const TCHAR *exp);
typedef TCalc (__stdcall* TCalcFor)(TCalc *args);
typedef bool (__stdcall* TAddFunction)(const TCHAR *exp,
MATHFUNCTION_PTR);
typedef bool (__stdcall* TRemoveFunction)(const T CHAR *exp);
typedef bool (__stdcall* TAddConst)(const TCHAR *exp, TCalc val);
typedef bool (__stdcall* TRemoveConst)(const TCHAR *exp);
typedef void (__stdcall* TSetErrorHandle)(TpfErrorHandler);



//External variables
extern  TEvaluateEqu        EvaluateEqu;
extern  TEvaluate           Evaluate;
extern  TBuildEqu                   BuildEqu;
extern  TBuild              Build;
extern  TCalcFor            CalcFor;
extern  TAddFunction        AddFunction;
extern  TRemoveFunction     RemoveFunction;
extern  TAddConst           AddConst;
extern  TRemoveConst        RemoveConst;
extern  TSetErrorHandle     SetErrorHandle;


//Load & UnLoad EquTranslator.dll
bool LoadEquTranslator(void);
void UnLoadEquTranslator(void);


////////////////////////////////////////////////////////////
//Exception class
class EquException {
        typedef std::basic_string<TCHAR>            STRING;
        const STRING err;
public:
        EquException(const TCHAR* _err): err(_err) {}
        EquException(const TCHAR* err1, const TCHAR * err2)
                : err( STRING(err1)+STRING(err2)) {}
        const STRING& what(void)const { return err; }
};

#endif
```

Also, source file EquTranslatorEx.cpp, which comes with package too, should be included in project:

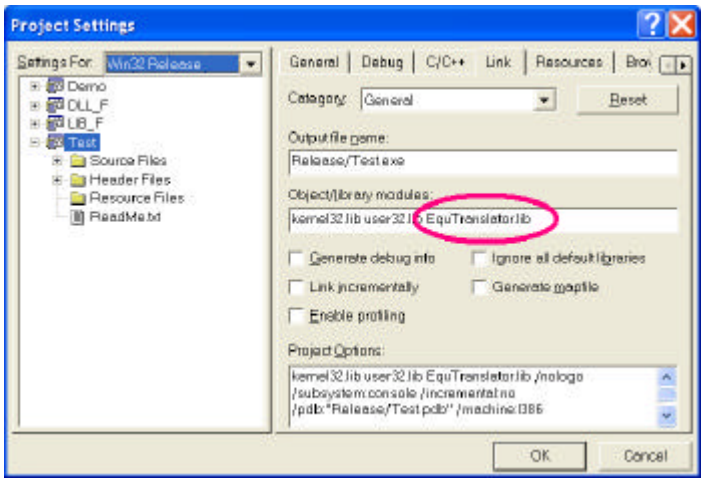
```
//EquTranslatorEx.cpp
//Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
```

```
//http://www.e-MathSolution.com
//For details contact us: info@e-MathSolution.com

#include "EquTranslatorEx.h"


TEvaluateEqu          EvaluateEqu;
TEvaluate             Evaluate;
TBuildEqu             BuildEqu;
TBuild                Build;
TCalcFor              CalcFor;
TAddFunction          AddFunction;
TRemoveFunction       RemoveFunction;
TAddConst             AddConst;
TRemoveConst          RemoveConst;
TSetErrorHandle       SetErrorHandle;
HINSTANCE hDLL=NULL;


#if defined(__BORLANDC__)
void __stdcall ErrorHandler(const TCHAR*err) throw( EquException)
{
    throw EquException(err);
}
#endif


bool LoadEquTranslator(void)
{
#if defined(_UNICODE)
    if((hDLL = LoadLibrary(_T("EquTranslatorW.dll")))==NULL)
#else
    if((hDLL = LoadLibrary(_T("EquTranslator.dll")))==NULL)
#endif
        return false;

    EvaluateEqu=(TEvaluateEqu)GetProcAddress(hDLL,_T("EvaluateEqu"));
    Evaluate=(TEvaluate)GetProcAddress(hDLL,_T("Evaluate"));
    BuildEqu=(TBuildEqu)GetProcAddress(hDLL,_T("BuildEqu"));
    Build=(TBuild)GetProcAddress(hDLL,_T("Build"));
    CalcFor=(TCalcFor)GetProcAddress(hDLL,_T("CalcFor"));
    AddFunction=(TAddFunction)GetProcAddress(hDLL,_T("AddFunction"));
     RemoveFunction=
        (TRemoveFunction)GetProcAddress(hDLL,_T("RemoveFunction"));
    AddConst=(TAddConst)GetProcAddress(hDLL,_T("AddConst"));

RemoveConst=(TRemoveConst)GetProcAddress(hDLL,_T("RemoveConst"));

SetErrorHandle=(TSetErrorHandle)GetProcAddress(hDLL,_T("SetErrorHandle"));

    if( !EvaluateEqu ||  !Evaluate ||   !BuildEqu ||   !Build ||
      !CalcFor || !AddFunction || !RemoveFunction || !AddConst ||
      !RemoveConst || !SetErrorHandle)
      return false;

#if defined(__BORLANDC__)
    SetErrorHandle(ErrorHandler);
#endif

    return true;
}

//////////////////////////////////////////////////////////////
//
void UnLoadEquTranslator(void)
{
    if(hDLL){
        SetErrorHandle(NULL);
      FreeLibrary(hDLL);
        }
}
```

Calling function looks like:

```
#include <iostream.h>
#include"EquTranslatorEx.h"


//User defined function
const TCalc __stdcall toRad(const TCalc& x)
{
        return x*3.14/180.;
}


int _tmain(int argc, _TCHAR* argv[])
{
        TCalc args[]={5, 1.5,-3.2};
        TCalc  res;
        TCalc twoPi=6.28;     //user defined const

        //Math expressions
   TCHAR *mathFunc = _T("F(x, y, z) = 3*x^2-y^2+sqrt(toRad(45))/twoPi");
   TCHAR *mathExp = _T("3*x^2-y^2+cos(toRad(85))/4.45+twoPi");
   TCHAR *varList = _T("x y z");

        if(!LoadEquTranslator())
            exit(1);

        try{
                AddFunction(_T("toRad"), toRad);
                AddConst(_T("twoPi"), twoPi);

            BuildEqu(mathFunc);
                res = CalcFor(args);
                cout<<"Res="<<res<<endl;

                RemoveFunction(_T("toRad"));
                Build(_T(""), _T("twoPi/2"));
                res = CalcFor(args);
            cout<<"Res="<<res<<endl;

                AddFunction(_T("toRad"), toRad);
                res = EvaluateEqu(mathFunc, args);
                cout<<"Res="<<res<<endl;

                RemoveConst(_T("twoPi"));
                res=Evaluate(varList, mathExp, args); //Error.
                cout<<"Res="<<res<<endl;

        } catch(EquException &e) {
                std::cout<<e.what().c_str();
                }

    UnLoadEquTranslator();
        return 0;
}
```

In Visual C++ is also possible implicit call of *EquTranslator*. In this case, Project/Settings must include EquTranslator.lib or EquTranslatorW.lib for UNICODE version (See figure below):



and programmer must include head file #include "EquTranslator.h" instead of "EquTranslatorEx.h" .

```
#ifndef EQUTRANSLATOR_H
#define EQUTRANSLATOR_H
#pragma warning(disable:4786)

#include <string>
#include <TCHAR.H>

//Type definition
typedef double TCalc;
//CallBack function to handle the errors
typedef void (__stdcall*TpfErrorHandler)(const TCHAR*);
//Math functions. User defined functions
typedef const TCalc (__stdcall *MATHFUNCTION_PTR)(const TCalc &);

////////////////////////////////////////////////////////////
// Interface
TCalc __stdcall EvaluateEqu(const TCHAR *exp, const TCalc *args );
TCalc __stdcall Evaluate(const TCHAR *arg, const TCHAR *exp, const TCalc
            *args );

void __stdcall  BuildEqu(const TCHAR *exp);
void __stdcall Build(const TCHAR *arg, const TCHAR *exp);
TCalc __stdcall CalcFor(const TCalc *args);

bool __stdcall AddFunction(const TCHAR*exp, MATHFUNCTION_PTR);
bool __stdcall RemoveFunction(const TCHAR*exp);
bool __stdcall AddConst(const TCHAR*exp, TCalc);
bool __stdcall RemoveConst(const TCHAR*exp);

 void __stdcall SetErrorHandle(TpfErrorHandler);

 void __stdcall InitEquTranslator(void);
 void __stdcall CloseEquTranslator(void);


////////////////////////////////////////////////////////////
//Exception class//////////////////////////////////////////////
class EquException {
            typedef std::basic_string<TCHAR>  STRING;
            const STRING err;
public:
            EquException(const TCHAR* _err): err(_err) {}
            EquException(const TCHAR* err1, const TCHAR * err2)
                        : err( STRING(err1)+STRING(err2)) {}
            const STRING& what(void)const { return err; }
};
#endif
```

Example 2: Delphi

Within Delphi environment the component initialization is done by initialization section of the unit. Just include source file EquTranslatotr.pas in your project:

```
//EquTranslator.pas
//Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
//http://www.e-MathSolution.com
//For details contact us: info@e-MathSolution.com

unit EquTranslator;

interface

uses
  SysUtils;

type
      TpDbl =^double;
      TpFunction=^TFunction;
      TpFuncError=^TFuncErr;
      TFunction=function(var ar:double):double;stdcall;
      TFuncErr=procedure(er:string);


//Interface of EquTranslator.dll
      function EvaluateEqu(exp:PChar; arg:TpDbl):double; stdcall;
      function Evaluate(varList:PChar; exp:PChar; arg:TpDbl):double; stdcall;
      procedure BuildEqu(exp: PChar);stdcall;
      procedure Build(varList:PChar; exp:PChar); stdcall;
      function CalcFor(arg:TpDbl):double; stdcall;
      function AddFunction(name:PChar;fn:TpFunction):boolean ;stdcall;
      function RemoveFunction(name:PChar):boolean ;stdcall;
      function AddConst(name:PChar;fn:double):boolean ;stdcall;
      function RemoveConst(name:PChar):boolean ;stdcall;
      procedure SetErrorHandle(errF:TpFuncError);stdcall;


implementation
      function EvaluateEqu; external 'Equtranslator.dll';
      function Evaluate; external 'Equtranslator.dll';
      procedure BuildEqu; external 'Equtranslator.dll';
      procedure Build; external 'Equtranslator.dll';
      function CalcFor; external 'Equtranslator.dll';
      function AddFunction; external 'Equtranslator.dll';
      function RemoveFunction; external 'Equtranslator.dll';
      function AddConst; external 'Equtranslator.dll';
      function RemoveConst; external 'Equtranslator.dll';
      procedure SetErrorHandle; external 'Equtranslator.dll';


//Native function to raise exception
procedure RiaseException(err:PChar);stdcall;
begin
      raise   Exception.Create(err);
end;


initialization
      //Init error handle
      SetErrorHandle( @RiaseException );
end.
```

Calling function looks like:

```
program Test;
{$APPTYPE CONSOLE}
uses
  SysUtils, EquTranslator, windows;

  //User defined function (grades to radians).
function toRad(var x:double):double; stdcall;
begin
      toRad:=x*3.14/180;
end;

var
  args:array[0..2] of double;
  res:double;
  twoPi:double;
  mathFunc:PChar;
  mathExp:PChar;
  varList:PChar;

 begin

 args[0]:=2;
 args[1]:=5.6;
 args[2]:=-11;
  twoPi:=6.28;

 mathFunc := 'F(x, y, z) = 3*x^2-y^2+sin(toRad(45))/twoPi';
 mathExp := '3*x^2-y^2+cos(toRad(85))/twoPi';
 varList := 'x y z';

   try
      AddFunction('toRad', @toRad);
      AddConst('twoPi', twoPi);

      BuildEqu(mathFunc);
              res := CalcFor(@args);
              writeln('res=',res);

              RemoveFunction('toRad');
              Build('', 'twoPi/2');
              res := CalcFor(@args);
              writeln('res=',res);

              AddFunction('toRad', @toRad);
              res := EvaluateEqu(mathFunc, @args);
              writeln('res=',res);

              RemoveConst('twoPi');
              res :=Evaluate(varList, mathExp, @args);//Error
              writeln('res=',res);


   except
       on E:Exception do writeln('Exception: '+e.Message);

      end;

  readln;
end.
```

Example 3: VB

Programmer should add to VB project source file EquTranslator.bas:

```
'EquTranslator.bas
'Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
'http://www.e-MathSolution.com
'For details contact us: info@e-MathSolution.com

'Init Interface
Declare Function Evaluate Lib "Equtranslator.dll" (ByVal argList$, ByVal expr$,
ByRef arg As Double) As Double
Declare Function EvaluateEqu Lib "Equtranslator.dll" (ByVal expr$, ByRef arg As
Double) As Double
Declare Sub BuildEqu Lib "Equtranslator.dll" (ByVal expr$)
Declare Sub Build Lib "Equtranslator.dll" (ByVal argList$, ByVal expr$)
Declare Function CalcFor Lib "Equtranslator.dll" (ByRef arg As Double) As Double

Declare Function AddFunction Lib "Equtranslator.dll" (ByVal constName$, ByVal
lpFunc As Long) As Boolean
Declare Function RemoveFunction Lib "Equtranslator.dll" (ByVal constName$) As
Boolean
Declare Function AddConst Lib "Equtranslator.dll" (ByVal constName$, ByVal arg
As Double) As Boolean
Declare Function RemoveConst Lib "Equtranslator.dll" (ByVal constName$) As
Boolean
Declare Sub SetErrorHandle Lib "Equtranslator.dll" (ByVal lpFunc As Long)



Public Declare Function strlen Lib "kernel32" Alias "lstrlenA" (ByVal lpStr As Any)
As Long
Public Declare Function strcpy Lib "kernel32" Alias "lstrcpyA" (ByVal lpStr1 As
String, ByVal lpStr2 As Long) As Long



Public Sub ErrorHndl(ByVal lPointer As Long)
   Dim lRetVal As Long
   Dim strErr As String

    strErr = String$(strlen(ByVal lPointer), 0)
    lRetVal = strcpy(ByVal strErr, ByVal lPointer)

    Err.Description = strErr
    Err.Raise 1
End Sub

'To convert function to pointer
Public Function FnPtrToLong(ByVal lngFnPtr As Long) As Long
  FnPtrToLong = lngFnPtr
End Function

'////////////////////////////////////////////////////////////////////////////////
'Uer defined function
Private Function qub(ByRef x As Double) As Double
   qub = x * x * x
End Function
```

Calling function looks like:

```
Private Sub Command1_Click()
Dim arg(3) As Double
On Error GoTo errHnd
   arg(0) = 3.6
   arg(1) = 5.6
   arg(2) = 15.6
   expr$ = "f(x,y,z)=sin(x)+x*(-y)+5.4789*exp(sin(x/(y)))-cos(z+3)/(y-p2*z)"
   p2 = 6.28

   SetErrorHandle FnPtrToLong(AddressOf ErrorHndl)
```

```
   AddFunction "qub", FnPtrToLong(AddressOf qub)

  AddConst "p2", p2

   expr$ = "f(x,y,z)=p2+(x+y)/sin(z)"
   BuildEqu expr$
   res = CalcFor(arg(0))
   Label1.Caption = Str(res)


   res = Evaluate("x,y,z", "x+y+z+p2+qub(x)", arg(0))
   Label2.Caption = Str(res)

   RemoveConst ("p2")
   res = EvaluateEqu("f(x1,x2)=qub(x1)/x2", arg(0))
   Label3.Caption = Str(res)

   Exit Sub

errHnd:   'Error
   Label3.Caption = Err.Description
 End Sub
```

Example 4: .NET

Also, *EquTranslator* algorithm can be called from a .NET application in the same way as it was described above. There are examples for C# and VB.NET. In case of C++.NET, please see C/C++ example.

C#

EquTranslator.cs should be included in a C# project. The file contains EMS namespace with EquTranlator and EquTranlatorW class:

```
//EquTranslator.cs
//Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
//http://www.e-MathSolution.com
//For details contact us: info@e-MathSolution.com
using System.Runtime.InteropServices;

namespace EMS
{
public class EquTranslator
{
   [DllImport("EquTranslator.dll",CallingConvention=CallingConvention.StdCall,Ch
            arSet=CharSet.Ansi)]
            public extern static  double EvaluateEqu(string exp, double [] arg);
   [DllImport("EquTranslator.dll",CallingConvention=CallingConvention.StdCall,
             CharSet=CharSet.Ansi)]
            public extern static double Evaluate(string arg, string exp, double []
                     argVal);

   [DllImport("EquTranslator.dll",CallingConvention=CallingConvention.StdCall,
             CharSet=CharSet.Ansi)]
            public extern static  void BuildEqu(string mExp);
   [DllImport("EquTranslator.dll",CallingConvention=CallingConvention.StdCall,
             CharSet=CharSet.Ansi)]
            public extern static  void Build(string arg, string exp);
   [DllImport("EquTranslator.dll",CallingConvention=CallingConvention.StdCall,
             CharSet=CharSet.Ansi)]
            public  extern static  double CalcFor(double [] arg);


public  delegate  double  CallBackUserFunction(ref double val);
   [DllImport("EquTranslator.dll",CallingConvention=CallingConvention.StdCall,
             CharSet=CharSet.Ansi)]
            public   extern static bool AddFunction(string
                     funName,CallBackUserFunction fn);
   [DllImport("EquTranslator.dll",CallingConvention=CallingConvention.StdCall,
             CharSet=CharSet.Ansi)]
            public extern static bool RemoveFunction(string exp);

   [DllImport("EquTranslator.dll",CallingConvention=CallingConvention.StdCall,
             CharSet=CharSet.Ansi)]
            public extern static bool AddConst(string exp, double val);
   [DllImport("EquTranslator.dll",CallingConvention=CallingConvention.StdCall,
             CharSet=CharSet.Ansi)]
            public extern static bool RemoveConst(string exp);


// Declaring Delegate for Callback Function
public delegate void CallBackDelegate(string err);
   [DllImport("EquTranslator.dll",CallingConvention=CallingConvention.StdCall,
             CharSet=CharSet.Ansi)]
            public static extern void  SetErrorHandle(CallBackDelegate dl);


public static void SetEquTranslatorException()
{
   CallBackDelegate errrBkCallFunc = new CallBackDelegate(RiseException);
   SetErrorHandle(errrBkCallFunc);
}

static void RiseException(string err)
{
            throw new System.Exception(err);
}
}

////////////////////////////////////////////////////////////
/////Unicode version of the EquTranslator algorithm
public class EquTranslatorW
{
[DllImport("EquTranslatorW.dll",CallingConvention=CallingConvention.StdCall,
             CharSet=CharSet.Unicode)]
            public extern static  double EvaluateEqu(string exp, double [] arg);
[DllImport("EquTranslatorW.dll",CallingConvention=CallingConvention.StdCall,
             CharSet=CharSet.Unicode)]
            public extern static double Evaluate(string arg, string exp, double []
                        argVal);

[DllImport("EquTranslatorW.dll",CallingConvention=CallingConvention.StdCall,
             CharSet=CharSet.Unicode)]
            public extern static  void BuildEqu(string mExp);
[DllImport("EquTranslatorW.dll",CallingConvention=CallingConvention.StdCall,
             CharSet=CharSet.Unicode)]
            public extern static  void Build(string arg, string exp);
[DllImport("EquTranslatorW.dll",CallingConvention=CallingConvention.StdCall,
             CharSet=CharSet.Unicode)]
            public  extern static  double CalcFor(double [] arg);


public  delegate  double  CallBackUserFunction(ref double val);
[DllImport("EquTranslatorW.dll",CallingConvention=CallingConvention.StdCall,
             CharSet=CharSet.Unicode)]
            public   extern static bool AddFunction(string
                        funName,CallBackUserFunction fn);
[DllImport("EquTranslatorW.dll",CallingConvention=CallingConvention.StdCall,
             CharSet=CharSet.Unicode)]
            public extern static bool RemoveFunction(string exp);

[DllImport("EquTranslatorW.dll",CallingConvention=CallingConvention.StdCall,
             CharSet=CharSet.Unicode)]
            public extern static bool AddConst(string exp, double val);
[DllImport("EquTranslatorW.dll",CallingConvention=CallingConvention.StdCall,
```

```
            CharSet=CharSet.Unicode)]
        public extern static bool RemoveConst(string exp);

// Declaring Delegate for Callback Function
public delegate void CallBackDelegate(string err);
[DllImport("EquTranslatorW.dll",CallingConvention=CallingConvention.StdCall,
            CharSet=CharSet.Unicode)]
        public static extern void  SetErrorHandle(CallBackDelegate dl);


public static void SetEquTranslatorException()
{
   CallBackDelegate errrBkCallFunc = new CallBackDelegate(RiseException);
   SetErrorHandle(errrBkCallFunc);
}

static void RiseException(string err)
{
        throw new System.Exception(err);
}
}
}
```

To set error handling mechanism just call function with delegate SetEquTranslatorException from EquTranslator(W) class. Example how to use Equtranslator exception in C# :

```
// Demo.cs :
//EquTranslator algorithm
//Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
//http://www.e-MathSolution.com
//For details contact us: info@e-MathSolution.com

using System;
using EMS;


namespace CSH_test
{
class Test
{
//User defined function (grades to radians).
static double toRad(ref double x)
{
        return 3.14*x/180;
}

[STAThread]
static void Main(string[] args)
{
    double [] arg= new double[2];
    double res=0;
    string mExp="f(x,y)=x+(5+y)^2-exp(x)+sin(toRad(x*y))";
    arg[0]=5; arg[1]=6;

    EMS.EquTranslator.CallBackUserFunction tRd=
                    new EMS.EquTranslator.CallBackUserFunction(toRad);
    try
    {
       //Set the error handle
       EMS.EquTranslator.SetEquTranslatorException();

       EMS.EquTranslator.AddConst("pi2", 6.28);
       EMS.EquTranslator.AddFunction("toRad", tRd);

       EMS.EquTranslator.BuildEqu("f(x,y)=sin(pi2/x)+tan(toRad(x-y))");
       res=EMS.EquTranslator.CalcFor(arg);
       Console.WriteLine("res={0}", res);

       Console.WriteLine("res={0}", EMS.EquTranslator.EvaluateEqu(mExp,
          arg));

       EMS.EquTranslator.RemoveConst("pi2");
       Console.WriteLine("res={0}",
          EMS.EquTranslator.Evaluate("x,y","cos(pi2/x)^(y/(5+x))", arg));
    }
    catch(Exception e)
    {
        Console.WriteLine(e.Message);
    }

}
}
}
```

VB.NET

EquTranslator.vb should be included in a VB.NET project. The file contains EMS namespace with EquTranlator and EquTranlatorW class:

```
'EquTranslator.vb
'Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
'http://www.e-MathSolution.com
'For details contact us: info@e-MathSolution.com

Imports System.Runtime.InteropServices
Namespace EMS
 Public Class EquTranslator
<DllImport("EquTranslator.dll", CallingConvention:=CallingConvention.StdCall,_
            CharSet:=CharSet.Ansi)> _
        Public Shared Function EvaluateEqu(ByVal exp As String, ByRef args As
            Double) As Double
    End Function
<DllImport("EquTranslator.dll", CallingConvention:=CallingConvention.StdCall,_
        CharSet:=CharSet.Ansi)> _
    Public Shared Function Evaluate(ByVal arg As String, ByVal exp As String,
        ByRef args As Double) As Double
    End Function
 <DllImport("EquTranslator.dll",CallingConvention:=CallingConvention.StdCall,_
            CharSet:=CharSet.Ansi)> _
     Public Shared Sub BuildEqu(ByVal arg As String)
    End Sub
<DllImport("EquTranslator.dll", CallingConvention:=CallingConvention.StdCall,_
        CharSet:=CharSet.Ansi)> _
    Public Shared Sub Build(ByVal arg As String, ByVal exp As String)
    End Sub
<DllImport("EquTranslator.dll", CallingConvention:=CallingConvention.StdCall,_
        CharSet:=CharSet.Ansi)> _
     Public Shared Function CalcFor(ByRef args As Double) As Double
    End Function


Public Delegate Function CallBackUserFunction(ByRef val As Double) As Double
<DllImport("EquTranslator.dll", CallingConvention:=CallingConvention.StdCall,_
        CharSet:=CharSet.Ansi)> _
    Public Shared Function AddFunction(ByVal fnName As String, ByVal lpFunc
        As CallBackUserFunction) As Boolean
    End Function
<DllImport("EquTranslator.dll", CallingConvention:=CallingConvention.StdCall,_
        CharSet:=CharSet.Ansi)> _
     Public Shared Function RemoveFunction(ByVal fnName As String) As
        Boolean
    End Function
```

```
<DllImport("EquTranslator.dll", CallingConvention:=CallingConvention.StdCall,_
        CharSet:=CharSet.Ansi)> _
    Public Shared Function AddConst(ByVal constName As String, ByVal val As
        Double) As Boolean
    End Function
<DllImport("EquTranslator.dll", CallingConvention:=CallingConvention.StdCall,_
        CharSet:=CharSet.Ansi)> _
     Public Shared Function RemoveConst(ByVal constName As String) As
        Boolean
    End Function

Public Delegate Sub CallBackDelegate(ByVal constName As String)
<DllImport("EquTranslator.dll", CallingConvention:=CallingConvention.StdCall,_
        CharSet:=CharSet.Ansi)> _
    Public Shared Sub SetErrorHandle(ByVal errDgt As CallBackDelegate)
    End Sub


    Public Sub SetEquTranslatorException()
      Dim errD As CallBackDelegate
      errD = New CallBackDelegate(AddressOf RiseException)
      SetErrorHandle(errD)
    End Sub

    Private Sub RiseException(ByVal err As String)
      Throw New Exception(err)
    End Sub

End Class


  '////////////////////////////////////////////////////////////////////////
  '//The UNICODE implementation of the EquTranslator algorithm
Public Class EquTranslatorW
<DllImport("EquTranslatorW.dll", CallingConvention:=CallingConvention.StdCall, _
          CharSet:=CharSet.Unicode)> _
        Public Shared Function EvaluateEqu(ByVal exp As String, ByRef args
As Double) As Double
    End Function
 <DllImport("EquTranslatorW.dll", CallingConvention:=CallingConvention.StdCall,_
        CharSet:=CharSet.Unicode)> _
    Public Shared Function Evaluate(ByVal arg As String, ByVal exp As String,
        ByRef args As Double) As Double
    End Function
<DllImport("EquTranslatorW.dll", CallingConvention:=CallingConvention.StdCall, _
        CharSet:=CharSet.Unicode)> _
    Public Shared Sub BuildEqu(ByVal arg As String)
    End Sub
<DllImport("EquTranslatorW.dll", CallingConvention:=CallingConvention.StdCall, _
        CharSet:=CharSet.Unicode)> _
    Public Shared Sub Build(ByVal arg As String, ByVal exp As String)
    End Sub
<DllImport("EquTranslatorW.dll", CallingConvention:=CallingConvention.StdCall, _
        CharSet:=CharSet.Unicode)> _
    Public Shared Function CalcFor(ByRef args As Double) As Double
    End Function


Public Delegate Function CallBackUserFunction(ByRef val As Double) As Double
<DllImport("EquTranslatorW.dll", CallingConvention:=CallingConvention.StdCall, _
        CharSet:=CharSet.Unicode)> _
     Public Shared Function AddFunction(ByVal fnName As String, ByVal lpFunc
        As CallBackUserFunction) As Boolean
    End Function
<DllImport("EquTranslatorW.dll", CallingConvention:=CallingConvention.StdCall, _
        CharSet:=CharSet.Unicode)> _
    Public Shared Function RemoveFunction(ByVal fnName As String) As
        Boolean
    End Function

<DllImport("EquTranslatorW.dll", CallingConvention:=CallingConvention.StdCall, _
        CharSet:=CharSet.Unicode)> _
    Public Shared Function AddConst(ByVal constName As String, ByVal val As
        Double) As Boolean
    End Function
 <DllImport("EquTranslatorW.dll", CallingConvention:=CallingConvention.StdCall, _
        CharSet:=CharSet.Unicode)> _
     Public Shared Function RemoveConst(ByVal constName As String) As
        Boolean
    End Function

Public Delegate Sub CallBackDelegate(ByVal constName As String)
<DllImport("EquTranslatorW.dll", CallingConvention:=CallingConvention.StdCall, _
        CharSet:=CharSet.Unicode)> _
    Public Shared Sub SetErrorHandle(ByVal errDgt As CallBackDelegate)
    End Sub


    Public Sub SetEquTranslatorException()
      Dim errD As CallBackDelegate
      errD = New CallBackDelegate(AddressOf RiseException)
      SetErrorHandle(errD)
    End Sub

    Private Sub RiseException(ByVal err As String)
      Throw New Exception(err)
    End Sub

  End Class
End Namespace
```

To set error handling mechanism just call the function with delegate SetEquTranslatorException from EquTranslator(W) class. Example how to use Equtranslator exception in VB.NET :

```
' Demo.vb :
'EquTranslator algorithm
'Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
'http://www.e-MathSolution.com
'For details contact us: info@e-MathSolution.com

Module Module1
  Sub Main()
    On Error GoTo errHnd
    Dim expr As String
    Dim arg(3) As Double
    Dim res As Double
    Dim tr As EMS.EquTranslator
    Dim qb As EMS.EquTranslator.CallBackUserFunction

    tr = New EMS.EquTranslator()
    arg(0) = 3
    arg(1) = 5.6
    arg(2) = 15.6
    expr = "f(x,y,z)=sin(x)+x*(-y)+5.4789*exp(sin(x/(y)))-cos(z+3)/(y-x*z)"

    qb = New EMS.EquTranslator.CallBackUserFunction(AddressOf qub)
    tr.AddFunction("qub", qb)

    tr.SetEquTranslatorException()

    res = tr.EvaluateEqu(expr, arg(0))
    System.Console.WriteLine("res={0}", res)
```

```
    res = tr.Evaluate("x,y", "qub(x)+exp(y)", arg(0))
    System.Console.WriteLine("res={0}", res)

errHnd:  'Error
    System.Console.WriteLine(Err.Description)

  End Sub

  'User defined function
  Function qub(ByRef x As Double) As Double
    qub = x * x * x
  End Function
End Module
```

Demo of the algorithm

Demo for EquTranslator algorithm is distributed as a stand-alone application *demo.exe* and EquTranslator.dll component, which can be used in your own projects.

Demo.exe

The Demo for *EquTranslator* algorithm is presented as console application demo.exe, which is built up on EquTr.lib.PRC version and it gives you main information how to use *EquTranslator* and about calculation speed.
Running the Demo, you will discover flexibility, power, and in the same time friendly and easy in use interface. And for sure, you will reveal parse-calculation SPEED. For this reason in the Demo was embedded code to print the start and end calculation time, also CPU ticks, so not only you can feel it but also you can see it.
The application is very simple and does not require the additional comments. Just run it and find out if your parser-calculator needs some improvements.

```
// Demo.cpp :
//EquTranslator algorithm
//Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
//http://www.e-MathSolution.com
//For details contact us: info@e-MathSolution.com

#include "stdafx.h"
#include <time.h>
#include <iostream.h>
#include <windows.h>

#include"EquTranslator.h"

int main(int argc, char* argv[])
{
        TCalc res=0;
        TCalc *args=NULL;
        clock_t start, finish;
        TCHAR funcExp[1000];
        long N, iterNum;
        SYSTEMTIME sysTimeStart, sysTimeEnd;

cout<<"EquTranslator.The Demo application."<<endl;
cout<<"Copyright © 2001-2005 Easy Math Solution. All Rights Reserved"<<endl;
cout<<"http://www.e-MathSolution.com"<<endl;
cout<<"For details contact us: info@e-MathSolution.com"<<endl<<endl;

cout<<"Please, enter function expression as F(x1, x2,...,xn)=f(x1,
x2,...,xn)."<<endl;
cout<<"Example:  F(x,y,z)=sin(x)-x+y^z"<<endl;
cin>>funcExp;

cout<<endl<<"Enter number of the arguments.  N=";
cin>>N;

        args=new TCalc[N];
        for(int j=0; j<N; j++) {
                cout<<"args["<<j+1<<"]=";
                cin>>args[j];
        }

cout<<endl<<"Enter number of iterations iterNum=";
cin>>iterNum;
cout<<endl<<"Please wait..."<<endl;

        GetLocalTime(&sysTimeStart);
        InitEquTranslator();

try{
        start = clock();

        BuildEqu(funcExp);
        for(long i=0; i<iterNum; i++)
                res+=CalcFor(args);
}
catch( EquException & e)
{
        cout<<e.what().c_str();
}
        finish = clock();
        GetLocalTime(&sysTimeEnd);
cout<<endl<<"=============================================="<<endl;
cout<<"Start time: "<<sysTimeStart.wMinute<<"(min)
"<<sysTimeStart.wSecond<<"(s) ";
cout<<sysTimeStart.wMilliseconds<<"(ms)"<<endl;
        GetLocalTime(&sysTimeEnd);

cout<<endl<<"For equation "<<funcExp<<", and "<<iterNum<<" iterations"<<endl;
cout<<"Processor time  "<<finish-start<<endl;
cout<<endl<<"Result="<<res<<endl;

cout<<"End time: "<<sysTimeEnd.wMinute<<"(min)
"<<sysTimeEnd.wSecond<<"(s) ";
cout<<sysTimeEnd.wMilliseconds<<"(ms)"<<endl;
cout<<"================================================"<<endl;

        if(args) delete[] args;
        CloseEquTranslator();
        return 0;
}
```

EquTranslator.dll

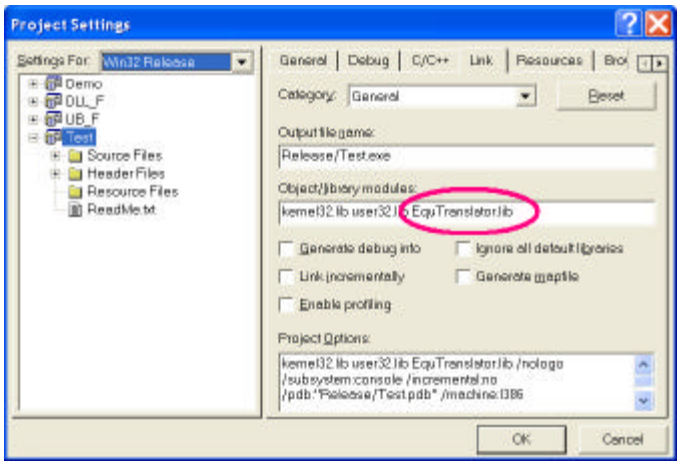
Also, *EquTranslator.zip* includes a demo version of the algorithm as *EquTranslator.dll* component and some project examples in VC++, C++Builder, VB, Delphi, C#, and VB.NET.
The *EquTranslator.dll* must be in active folder of the calling application to run properly. And, the demo version has been limited by 2 build-in functions (sqrt(), exp()) and only 1 user-defined function allowed. These restrictions are defined for the demo version only.
If some questions arise, please, don't hesitate and ask us about:
support@e-MathSolution.com

Appendix A: Examples

Visual C++

EquTr.lib.PRC

EquTr.lib.PRC is very easy to use. Programmer must include head file #include "EquTranslator.h" and add EquTranslator.lib or EquTranslatorW.lib as appropriate in Project Setting window (see figure below). The functions InitEquTranslator( ) and CloseEquTranslator() must be called explicitly.




```
// Test.cpp :
//EquTranslator algorithm
//Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
//http://www.e-MathSolution.com
//For details contact us: info@e-MathSolution.com

#include <time.h>
#include <iostream.h>
#include <TCHAR.h>

#include"EquTranslator.h"


const TCalc __stdcall toRad(const TCalc& x)
{
        return x*3.14/180;
}
int main(int argc, char* argv[])
{
    TCalc args[] = {2, 5.6, -11};
    TCalc res,twoPi=6.28;
    TCHAR *mathFunc = _T("F(x, y, z) = 3*x^2-y^2+sin(toRad(45))/twoPi");
    TCHAR *mathExp = _T("3*x^2-y^2+cos(toRad(85))/4");
    TCHAR *varList = _T("x y z");

   InitEquTranslator();

    try{
AddFunction(_T("toRad"), toRad);
AddConst(_T("twoPi"), twoPi);

BuildEqu(mathFunc);
        res = CalcFor(args);
        cout<<"Res="<<res<<endl;

        RemoveFunction(_T("toRad"));
        Build(_T(""), _T("twoPi/2"));
        res = CalcFor(args);
        cout<<"Res="<<res<<endl;

        AddFunction(_T("toRad"), toRad);
        res = EvaluateEqu(mathFunc, args);
        cout<<"Res="<<res<<endl;

        RemoveConst(_T("twoPi"));
//error "Uknown function: twoPi" occurs
        res=Evaluate(_T("f(x;y;z)"), _T("(x+y)*z+twoPi"), args);
        cout<<"Res="<<res<<endl;

    } catch(EquException &e) {
        cout<<e.what().c_str();
    }

    CloseEquTranslator();
     return 0;
}
```

EquTr.lib.OOP

The Object Oriented version of ***EquTranslator*** has some advantage over procedural. Programmer can create unlimited number of the objects to set parallel calculation and can pass the objects to the functions as argument by reference. Programmer should include head file #include "EquTranslator.h" and add EquTranslator.lib or EquTranslatorW.lib correspondingly to the Project Setting window (see EquTr.lib.PRC explanation above). EquTr.lib.OOP requires only InitEquTranslator() to be called.

```
// Test.cpp :
//EquTranslator algorithm
//Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
//http://www.e-MathSolution.com
//For details contact us: info@e-MathSolution.com

#include <time.h>
#include <iostream.h>
#include <TCHAR.h>

#include"EquTranslator.h"


const TCalc __stdcall toRad(const TCalc& x)
{
        return x*3.14/180;
}

TCalc PassObject(EquTranslator &p)
{
        TCalc x[]={3,5,8.9};
        return p.CalcFor(x);
}

int main(int argc, char* argv[])
{
    TCalc args[] = {2, 5.6, -11};
     TCalc res, twoPi=6.28;
     TCHAR *mathFunc = _T("F(x, y, z) = 3*x^2-y^2+sin(toRad(45))/twoPi");
     TCHAR *mathExp = _T("3*x^2-y^2+cos(toRad(85))/4");
     TCHAR *varList = _T("x y z");
        EquTranslator tr1, tr2;

        EquTranslator::InitEquTranslator();
```

```
	try{
		tr1.AddFunction(_T("toRad"), toRad);
		tr1.AddConst(_T("twoPi"), twoPi);

		tr1.BuildEqu(mathFunc);
		res = tr1.CalcFor(args);
		cout<<"Res="<<res<<endl;

		tr2.RemoveFunction(_T("toRad"));
		tr2.Build(_T(""), _T("twoPi/2"));
		res = tr2.CalcFor(args);
		 cout<<"Res="<<res<<endl;

		tr1.AddFunction(_T("toRad"), toRad);
		res = tr1.EvaluateEqu(mathFunc, args);
		cout<<"Res="<<res<<endl;

		res=tr1.Evaluate(_T("f(x;y;z)"), _T("(x+y)*z+twoPi"), args);
		cout<<"Res="<<res<<endl;

		tr2.RemoveConst(_T("twoPi"));
		tr1.Build(varList, mathExp);
		cout<<"Res="<<PassObject(tr1)<<endl;

	} catch(EquException &e) {
		cout<<e.what().c_str();
	}

	return 0;
}
```

The user defined functions and constants are global. It's meen that any object of *EquTranslator* has access to the same functions and constants.

EquTr.dll.PRC

In this type of implementation it is possible to use EquTranslator(W).dll in explicit and implicit way. The implicit call requires to include head file #include "EquTranslator.h" and add EquTranslator.lib or EquTranslatorW.lib as appropriate in Project Setting window (see EquTr.lib.PRC explanation above) and do not require initialization.

```
// Test.cpp :
//EquTranslator algorithm
//Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
//http://www.e-MathSolution.com
//For details contact us: info@e-MathSolution.com

#include <time.h>
#include <iostream.h>
#include <TCHAR.h>

#include"EquTranslator.h"


const TCalc __stdcall toRad(const TCalc& x)
{
	return x*3.14/180;
}

int main(int argc, char* argv[])
{
	TCalc args[] = {2, 5.6, -11};
	TCalc res,twoPi=6.28;
	TCHAR *mathFunc = _T("F(x, y, z) = 3*x^2-y^2+sin(toRad(45))/twoPi");
	TSHAR *mathExp = _T("3*x^2-y^2+cos(toRad(85))/4");
	TCHAR *varList = _T("x y z");

	try{
AddFunction(_T("toRad"), toRad);
AddConst(_T("twoPi"), twoPi);

BuildEqu(mathFunc);
		res = CalcFor(args);
		cout<<"Res="<<res<<endl;

		RemoveFunction(_T("toRad"));
		Build(_T(""), _T("twoPi/2"));
		res = CalcFor(args);
		cout<<"Res="<<res<<endl;

		AddFunction(_T("toRad"), toRad);
		res = EvaluateEqu(mathFunc, args);
		cout<<"Res="<<res<<endl;

		RemoveConst(_T("twoPi"));
		res=Evaluate(_T("f(x;y;z)"), _T("(x+y)*z+twoPi"), args);  //error "Uknown
function: twoPi" occurs
		cout<<"Res="<<res<<endl;

	} catch(EquException &e) {
		cout<<e.what().c_str();
	}

	 return 0;
}
```

But, in explicit call case, it's required to include head file #include "EquTranslatorEx.h" and to add to the project source file EquTranslatorEx.cpp. Text for EquTranslatorEx.h and EquTranslatorEx.cpp is shown in "Initialization and Error Handling" section Example1: C/C++. Sample below shows how to use it. Two function LoadEquTranslator() and UnLoadEquTranslator() are involved in initialization process.

```
// Test.cpp :
//EquTranslator algorithm
//Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
//http://www.e-MathSolution.com
//For details contact us: info@e-MathSolution.com

#include <time.h>
#include <iostream.h>
#include <TCHAR.h>

#include"EquTranslatorEx.h"


const TCalc __stdcall toRad(const TCalc& x)
{
	return x*3.14/180;
}

int main(int argc, char* argv[])
{
	TCalc args[] = {2, 5.6, -11};
	TCalc res,twoPi=6.28;
	TCAHR *mathFunc = _T("F(x, y, z) = 3*x^2-y^2+sin(toRad(45))/twoPi");
	TCHAR *mathExp = _T("3*x^2-y^2+cos(toRad(85))/4");
	TCHAR *varList = _T("x y z");

	if(!LoadEquTranslator())
		exit(0);

	 try{
```

```
        AddFunction(_T("toRad"), toRad);
        AddConst(_T("twoPi"), twoPi);

        BuildEqu(mathFunc);
        res = CalcFor(args);
        cout<<"Res="<<res<<endl;

        RemoveFunction(_T("toRad"));
        Build(_T(""), _T("twoPi/2"));
        res = CalcFor(args);
        cout<<"Res="<<res<<endl;

        AddFunction(_T("toRad"), toRad);
        res = EvaluateEqu(mathFunc, args);
        cout<<"Res="<<res<<endl;

        RemoveConst(_T("twoPi"));
        res=Evaluate(_T("f(x;y;z)"), _T("(x+y)*z+twoPi"), args);  //error "Uknown
function: twoPi" occurs
        cout<<"Res="<<res<<endl;

    } catch(EquException &e) {
        cout<<e.what().c_str();
    }
    UnLoadEquTranslator();

     return 0;
}
```

EquTr.dll.OOP

Programmer should include head file #include “EquTranslator.h” and add EquTranslator.lib or EquTranslatorW.lib as appropriate in Project Setting window (see EquTr.lib.PRC explanation above). Iinitialization is not required.

```
// Test.cpp :
//EquTranslator algorithm
//Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
//http://www.e-MathSolution.com
//For details contact us: info@e-MathSolution.com

#include <time.h>
#include <iostream.h>
#include <TCHAR.h>

#include"EquTranslator.h"


const TCalc __stdcall toRad(const TCalc& x)
{
        return x*3.14/180;
}

TCalc PassObject(EquTranslator &p)
{
        TCalc x[]={3,5,8.9};

        return p.CalcFor(x);
}

int main(int argc, char* argv[])
{
Calc args[] = {2, 5.6, -11};
TCalc res,twoPi=6.28;
TCHAR *mathFunc = _T("F(x, y, z) = 3*x^2-y^2+sin(toRad(45))/twoPi");
TCHAR *mathExp = _T("3*x^2-y^2+cos(toRad(85))/4");
TCHAR *varList = _T("x y z");
        EquTranslator tr1, tr2;

        EquTranslator.AddFunction(_T("toRad"), toRad);
try{
        tr1.AddConst(_T("twoPi"), twoPi);

                tr1.BuildEqu(mathFunc);
        res = tr1.CalcFor(args);
        cout<<"Res="<<res<<endl;

                tr2.RemoveFunction(_T("toRad"));
        tr2.Build(_T(""), _T("twoPi/2"));
        res = tr2.CalcFor(args);
         cout<<"Res="<<res<<endl;

        tr1.AddFunction(_T("toRad"), toRad);
        res = tr1.EvaluateEqu(mathFunc, args);
        cout<<"Res="<<res<<endl;

        res=tr1.Evaluate(_T("f(x;y;z)"), _T("(x+y)*z+twoPi"), args);
        cout<<"Res="<<res<<endl;

        tr2.RemoveConst(_T("twoPi"));
        tr1.Build(varList, mathExp);
        cout<<"Res="<<PassObject(tr1)<<endl;

} catch(EquException &e) {
        cout<<e.what().c_str();
}

return 0;
}
```

The user defined functions and constants are global. It’s meen that any object of *EquTranslator* has access to the same functions and constants.

C++Builder

EquTr.dll.PRC

The same as for VC++, explicit call.

Delphi

EquTr.dll.PRC

Within Delphi environment the component initialization is done by initialization section of the unit. Just include source file EquTranslatotr.pas (see source at “Initialization and Error Handling” section Example2: Delphi) in your project. The example how to use *EquTranslator* in Delphi is shown below:

```
program Test;
{$APPTYPE CONSOLE}
uses
  SysUtils, EquTranslator, windows;

function toRad(var x:double):double; stdcall;
begin
     toRad:=x*3.14/180;
end;

var
  args:array[0..2] of double;
  res:double;
  twoPi:double;
```

```
 mathFunc:PChar;
 mathExp:PChar;
 varList:PChar;

begin

args[0]:=2;
args[1]:=5.6;
args[2]:=-11;
 twoPi:=6.28;

 mathFunc := 'F(x, y, z) = 3*x^2-y^2+sin(toRad(45))/twoPi';
 mathExp := '3*x^2-y^2+cos(toRad(85))/4';
 varList := 'x y z';

  try
    AddFunction('toRad', @toRad);
    AddConst('twoPi', twoPi);

    BuildEqu(mathFunc);
          res := CalcFor(@args);
          writeln('res=',res);

          RemoveFunction('toRad');
          Build('', 'twoPi/2');
          res := CalcFor(@args);
          writeln('res=',res);

          AddFunction('toRad', @toRad);
          res := EvaluateEqu(mathFunc, @args);
          writeln('res=',res);

          RemoveConst('twoPi');
//error "Uknown function: twoPi" occurs
          res :=Evaluate('f(x;y;z)', '(x+y)*z+twoPi', @args);
          writeln('res=',res);

  except
     on E:Exception do writeln('Exception: '+e.Message);
    end;

 readln;
end.
```

VB

EquTr.dll.PRC

To use the component in VB, just add EquTranslator.bas to VB project. The source code of EquTranslator.bas you can find at "Initialization and Error Handling" section Example3: VB. Calling function may look like:

```
'Test.bas
'Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
'http://www.e-MathSolution.com
'For details contact us: info@e-MathSolution.com
Private Sub Command1_Click()
Dim arg(3) As Double
On Error GoTo errHnd
  arg(0) = 3.6
  arg(1) = 5.6
  arg(2) = 15.6
  expr$ = "f(x,y,z)=sin(x)+x*(-y)+5.4789*exp(sin(x/(y)))-cos(z+3)/(y-p2*z)"
  p2 = 6.28

  SetErrorHandle FnPtrToLong(AddressOf ErrorHndl)
  AddFunction "qub", FnPtrToLong(AddressOf qub)
  AddConst "p2", p2

  expr$ = "f(x,y,z)=p2+(x+y)/sin(z)"
  BuildEqu expr$
  res = CalcFor(arg(0))
  Label1.Caption = Str(res)


  res = Evaluate("x,y,z", "x+y+z+p2+qub(x)", arg(0))
  Label2.Caption = Str(res)

  RemoveConst ("p2")
  res = EvaluateEqu("f(x1,x2)=qub(x1)/x2", arg(0))
  Label3.Caption = Str(res)

  Exit Sub

errHnd:  'Error
  Label3.Caption = Err.Description
End Sub
```

C#

EquTr.dll.PRC

To use the EquTranslator algorithm in C#, just add EquTranslator.cs to the project. The source code of EquTranslator.cs you can find at "Initialization and Error Handling" section Example 4: C#. The calling function may look like:

```
// Demo.cs :
//EquTranslator algorithm
//Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
//http://www.e-MathSolution.com
//For details contact us: info@e-MathSolution.com

using System;
using EMS;

namespace CSH_test
{
class Test
{
        //User defined function (grades to radians).
        static double toRad(ref double x)
        {
                return 3.14*x/180;
        }

[STAThread]
static void Main(string[] args)
{
        double [] arg= new double[2];
        double res=0;
        string mExp="f(x,y)=x+(5+y)^2-exp(x)+sin(toRad(x*y))";
        arg[0]=5; arg[1]=6;

        EMS.EquTranslator.CallBackUserFunction tRd=
                new EMS.EquTranslator.CallBackUserFunction(toRad);
try
{
        //Set the error handle
        EMS.EquTranslator.SetEquTranslatorException();
        EMS.EquTranslator.AddConst("pi2", 6.28);

        EMS.EquTranslator.AddFunction("toRad", tRd);

        EMS.EquTranslator.BuildEqu("f(x,y)=sin(pi2/x)+tan(toRad(x-y))");
```

```
            res=EMS.EquTranslator.CalcFor(arg);
            Console.WriteLine("res={0}", res);

            Console.WriteLine("res={0}",
                                EMS.EquTranslator.EvaluateEqu(mExp, arg));

            EMS.EquTranslator.RemoveConst("pi2");
            Console.WriteLine("res={0}",

            EMS.EquTranslator.Evaluate("x,y","cos(pi2/x)^(y/(5+x))", arg));
}
catch(Exception e)
{
            Console.WriteLine(e.Message);
}

}
}
}
```

VB.NET

EquTr.dll.PRC

To use the EquTranslator algorithm in VB.NET, just add EquTranslator.vb to the project. The source code of EquTranslator.vb you can find at "Initialization and Error Handling" section Example 4: VB.NET. The calling function may look like:

```
' Demo.vb :
'EquTranslator algorithm
'Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
'http://www.e-MathSolution.com
'For details contact us: info@e-MathSolution.com

Module Module1

  Sub Main()
    On Error GoTo errHnd
    Dim expr As String
    Dim arg(3) As Double
    Dim res As Double
    Dim tr As EMS.EquTranslator
    Dim qb As EMS.EquTranslator.CallBackUserFunction

    tr = New EMS.EquTranslator()
    arg(0) = 3
    arg(1) = 5.6
    arg(2) = 15.6
    expr = "f(x,y,z)=sin(x)+x*(-y)+5.4789*exp(sin(x/(y)))-cos(z+3)/(y-x*z)"

    qb = New EMS.EquTranslator.CallBackUserFunction(AddressOf qub)
    tr.AddFunction("qub", qb)

    tr.SetEquTranslatorException()

    res = tr.EvaluateEqu(expr, arg(0))
    System.Console.WriteLine("res={0}", res)

    res = tr.Evaluate("x,y", "qub(x)+exp(y)", arg(0))
    System.Console.WriteLine("res={0}", res)

errHnd:  'Error
    System.Console.WriteLine(Err.Description)

  End Sub

  'User defined function
  Function qub(ByRef x As Double) As Double
    qub = x * x * x
  End Function
End Module
```