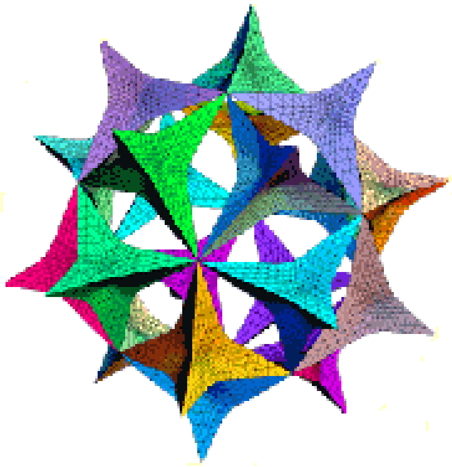


Easy *Math* Solution

`CxTranslator` (Version 1.1)

Context

Introduction

Thank you for your interest in ***Easy Math Solution*** products. *CxTranslator* is an equation parser-calculator for COMPLEX number math expressions with parse-tree builder and user-friendly interface for parsing and calculation a run-time defined a math complex numbers 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 and constants, which can be complex numbers (for designation image part of the complex numbers use ‘i’ or ‘j’ letter),
- operators: + - * / ^ @,
- functions abs, arg, conj, cos, cosh, exp, imag, log, log10, norm, real, sin, sinh, sqrt, tan, tanh,
- constants e= 2.71828182845904 and pi= 3.14159265358979, and
- user defined functions and constants.

For example:

*F(x, y) = 3i*x^2+y^2j+e/4i,*
F(x1, x2, y) = (x1-sin(x2))/y,
F()=5+6i+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 *CxTranslatorin* in C/C++, Delphi, VB, .NET and other languages and systems that can understand dll’s calls and COM.

The main characteristics of *CxTranslator*:

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

The full list of the build-in functions and arithmathic operators are shown in table below:

Functions

abs	Extracts the modulus of a complex number.
arg	Extracts the argument from a complex number. The argument is the angle that the complex vector makes with the positive real axis in the complex plane. For a complex number $a + bi$, the argument is equal to arctan(b/a). The angle has a positive sense when measured in a counterclockwise direction from the positive real axis and a negative sense when measured in a clockwise direction. The principal values are greater than -pi and less than or equal to +pi.
conj	Returns the complex conjugate of a complex number. The complex conjugate of a complex number $a + bi$ is $a - bi$. The product of a complex number and its conjugate is the norm of the number $a^2 + b^2$.
cos	Returns the cosine of a complex number.
cosh	Returns the hyperbolic cosine of a complex number.
exp	Returns the exponential function of a complex number.
imag	Extracts the imaginary component of a complex number.
log	Returns the natural logarithm of a complex number.
log10	Returns the base 10 logarithm of a complex number.
norm	Extracts the norm of a complex number. The norm of a complex number $a + bi$ is $(a^2 + b^2)$. The norm of a complex number is the square of its modulus. The modulus of a complex number is a measure of the length of the vector representing the complex number. The modulus of a complex number $a + bi$ is **sqrt**$(a^2 + b^2)$, written $\|a + bi\|$.
real	Extracts the real component of a complex number.
sin	Returns the sine of a complex number.
sinh	Returns the hyperbolic sine of a complex number.
sqrt	Returns the square root of a complex number.
tan	Returns the tangent of a complex number.
tanh	Returns the hyperbolic tangent of a complex number.

Operators

*	Multiplies two complex numbers, one or both of which may belong to the subset of the type for the real and imaginary parts.
+	Adds two complex numbers, one or both of which may belong to the subset of the type for the real and imaginary parts.
-	Subtracts two complex numbers, one or both of which may belong to the subset of the type for the real and imaginary parts.
/	Divides two complex numbers, one or both of which may belong to the subset of the type for the real and imaginary parts.
^	Evaluates the complex number obtained by raising a base that is a complex number to the power of another complex number.
@	Evaluate polar, which corresponds to a specified modulus and argument, in Cartesian form: $r@q$. The polar form of a complex number provides the modulus r and the argument q, where these parameters are related to the real and imaginary Cartesian components a and b by the equations $a = r * \cos(q)$ and $b = r * \sin(q)$. Example: 5@8. 5 is *Modulus* and 8 is *Argument.* @ has the same priority as + or -.

Try the DEMO version of *CxTranslator* 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, *CxTranslator* algorithm has different implementation. In this chapter we describe tags, which represent variety of the implementations, and type of the licenses.
So, *CxTranslator* 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 CxTr.
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, *CxTranslator* can be used for personal projects only. It covers software development process with *CxTranslator* 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

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

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

All prices are given in US dollars

Tag \ License	Iome	.cademic	rofessional	Use
CxTr.lib.PRC				VC++
CxTr.lib.OOP				VC++
CxTr.dll.PRC				VC++, Delphi, VB, C++Builder, .NET…
CxTr.dll.OOP				VC++

*To get the right price for *CxTranslator*, 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”: CxTranslatorW.lib or CxTranslatorW.dll
Want to get more information, please send request to license@e-MathSolution.com .

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 *CxTranslator* algorithm is a calculation type, which is defined as:

```
typedef std::complex<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[] = {TCalc(3.6,6),TCalc(5.6,-4),
                     TCalc(-1.5,0.33) };
     TCalc res;
     TCHAR *mathFunc = "F(x, y, z) = 3*x^2-y^2+z/4+sin(x+5.6i)";
……..
     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[] = {TCalc(3.6,6),TCalc(5.6,-4),
                  TCalc(-1.5,0.33) };
     TCalc res;
     TCHAR *mathExp = "3*x^2-y^2+z/4+sin(x+5.6i )";
     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[] = {TCalc(3.6,6),TCalc(5.6,-4),
                    TCalc(-1.5,0.33) };
  TCalc res;
  TCHAR *mathFunc = "F(x, y, z) = 3*x^2-y^2+z/4+sin(x+5.6i )";
  TCHAR *mathExp = "3j*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 CxTranslator 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 TCalc(x.real()*3.14/180,0);
}

int main(int argc, char* argv[])
{
TCalc args[] = {TCalc(3.6,6),TCalc(5.6,-4),
          TCalc(-1.5,0.33) };
TCalc res;
TCHAR *mathFunc = "F(x, y, z) = 3*x^2-y^2+sin(toRad(45))/4j";
TCHAR *mathExp = "3*x^2-y^2+cos(toRad(85))/4j";
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[]= {TCalc(3.6,6),TCalc(5.6,-4),
            TCalc(-1.5,0.33) };
  TCalc res;
  TCalc two_pi ( 2*3.14, 0);
  TCHAR *mathFunc = "F(x, y, z) = 3i*x^2-y^2+z/twoPi";
  TCHAR *mathExp = "3j*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 *CxTranslator* use. Some times initialization happens without programmer's awareness, but in most cases programmer should do some steps first to use *CxTranslator* for parse and calculations.
In general *CxTranslator* has two functions
void InitCxTranslator(void) and void CloseCxTranslator(void),
and, it depends on the implementation and programming approach, both, one or none functions may be called.
So, in case of:

- CxTr.lib.PRC - both function must be called: InitCxTranslator() and CloseCxTranslator();
- CxTr.lib.OOP – programmer must call InitCxTranslator() only;
- CxTr.dll.PRC – the implicit and explicit call of CxTranslator.dll involve InitCxTranslator() and CloseCxTranslator() automatically. Some programming languages and systems require including in project distributed with CxTranslator source modules that have subroutines to load dll and to initialize interface (see examples);
- CxTr.dll.OOP – to call InitCxTranslator() and CloseCxTranslator() 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. *CxTranslator* 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 *CxTranslator*, for example:

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

Calling `SetErrorHandle(RiaseException)`, programmer establishes mechanism to redirect all error messages from *CxTranslator* 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 CxTr.dll.PRC version of *CxTranslator* algorithm.

Example 1: C/C++

For explicit use of CxTranslator(W).dll, head file CxTranslatorEx.h must be included in projects:

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

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


//The type definitions
typedef double  CmpBaseType;
typedef std::complex<CmpBaseType> 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 TCHAR *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 CxTranslator.dll
bool LoadCxTranslator(void);
void UnLoadCxTranslator(void);


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

#endif
```

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

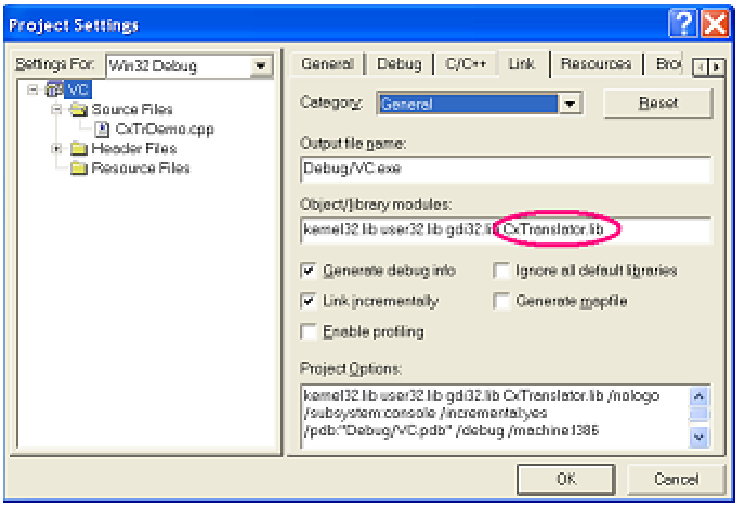
```
//CxTranslatorEx.cpp
//Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
//http://www.e-MathSolution.com
//For details contact us: info@e-MathSolution.com

#include "CxTranslatorEx.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( CxException)
{
    throw CxException(err);
}
#endif


bool LoadCxTranslator(void)
{
#if defined(_UNICODE)
    if((hDLL = LoadLibrary(_T("CxTranslatorW.dll")))==NULL)
#else
    if((hDLL = LoadLibrary(_T("CxTranslator.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 UnLoadCxTranslator(void)
{
        if(hDLL){
              SetErrorHandle(NULL);
             FreeLibrary(hDLL);
         }
}
```

Calling function looks like:

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


//User defined function
const TCalc __stdcall toRad(const TCalc& x)
{
              return TCalc(x.real()*3.14/180,0);
}

int main(int argc, char* argv[])
{
TCalc args[]={TCalc(3.6,6),TCalc(5.6,-4), TCalc(-1.5,0.33)};
TCalc res;
Atom *mathFunc = _T("F(x, y, z) = 3*x^2-y^2+sin(toRad(45))/4i");
Atom *mathExp = _T("3j*x^2-y^2+cos(toRad(85))/4");
Atom *varList = _T("x y z");

              if(!LoadCxTranslator())
              exit(0);

try{
              AddFunction(_T("toRad"), toRad);
              BuildEqu(mathFunc);
              res = CalcFor(args);
              cout<<"Res="<<res<<endl; //no errors

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

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

UnLoadCxTranslator();
              return 0;
}
```

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



Settings For: Win32 Debug

Category: General

Output file name: Debug/VC.exe

Object/library modules: kernel32.lib user32.lib gdi32.li CxTranslator.lib

☑ Generate debug info ☐ Ignore all default libraries

☑ Link incrementally ☐ Generate mapfile

☐ Enable profiling

Project Options:

```
kernel32.lib user32.lib gdi32.lib CxTranslator.lib /nologo
/subsystem:console /incremental:yes
/pdb:"Debug/VC.pdb" /debug /machine:I386
```

and programmer must include head file `#include "CxTranslator.h"` instead of `"CxTranslatorEx.h"`.

```
#ifndef CXTRANSLATOR_H
#define CXTRANSLATOR_H

#pragma warning(disable:4786)

#include <string>
#include <complex>
#include <TCHAR.h>

//Type definition
typedef double  CmpBaseType;
typedef std::complex<CmpBaseType> TCalc;

typedef void (__stdcall*TpfErrorHandler)(const TCHAR*);
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 InitCxTranslator(void);
             void __stdcall CloseCxTranslator(void);



//////////////////////////////////////////////////////////////////////
//Exception class/////////////////////////////////////////////////////
class CxException {
            typedef std::basic_string<TCHAR>            STRING;
            const STRING err;
public:
            CxException(const TCHAR* _err): err(_err) {}
            CxException(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 CxTranslatotr.pas in your project:

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

unit CxTranslator;

interface

uses
  SysUtils;

  type TComplex=record
  Re,Im:double;
  end;

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


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

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

  windows,
  CxTranslator in 'CxTranslator.pas';

function toRad(var x: TComplex):TComplex; stdcall;
var res:TComplex;
begin
        res.Im:=0;
        res.re:=x.re*3.14/180;
        toRad:=res;
end;

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

 begin

 args[0].Re:=3.6;args[0].Im:=6;
 args[1].Re:=5.6;args[1].Im:=-4;
 args[2].Re:=-1.5;args[2].Im:=0.33;
 twoPi.Re:=6.28;      twoPi.Im:=0;

  mathFunc := 'f(x,y,z)=sin(x)+x*(-y)+5.4789i*exp(sin(x/(toRad(y))))-cos(z+3)/(y-
                                          x^z)+twoPi';
  mathExp := '3*x^2-y^2+cos(85)/4i+twoPi';
  varList := 'x y z';
```

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

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

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

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

            RemoveConst('twoPi');
            res :=Evaluate(varList, mathExp, @args);
            writeln('Re=',res.Re,'    Im=',res.Im);


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

       end;

  readln;
 end.
```

Example 3: VB

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

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

Type Complex
   re As Double
   im As Double
End Type

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

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


Public Function FnPtrToLong(ByVal lngFnPtr As Long) As Long
  FnPtrToLong = lngFnPtr
End Function

Public Function qub(ByRef x As Complex) As Complex
   qub.re = x.re * x.re * x.re
End Function
```

Calling function looks like:

```
Private Sub Command1_Click()
On Error GoTo errHnd
   Dim arg(3) As Complex
    Dim res As Complex


   On Error GoTo errHnd
   arg(0).re = 3.6
   arg(0).im = 6
   arg(1).re = 5.6
   arg(1).im = -4
      arg(2).re = -1.5
      arg(2).im = 0.33

   expr$ = "f(x,y,z)=sin(x)+qub(x*(-y))+5.4789i*exp(sin(x/(y)))-cos(z+3)"

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

   BuildEqu expr$
   res = CalcFor(arg(0))
   Label1.Caption = Str(res.re)
   Label2.Caption = Str(res.im)

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

   Exit Sub

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

Example 4: .NET

Also, *CxTranslator* 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#

CxTranslator.cs should be included in a C# project. The file contains EMS namespace with CxTranlator and CxTranlatorW class:

```
//CxTranslator.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
{
[StructLayoutAttribute(LayoutKind.Sequential,
        CharSet=CharSet.Auto)]
        public struct Complex
                {
                public double re;
                public double im;
                }

public class CxTranslator
        {
//EvaluateEqu
[DllImport("CxTranslator.dll",CallingConvention=CallingConvention.StdCall,
                        CharSet=CharSet.Ansi)]
        public extern static  Complex EvaluateEqu(string exp, Complex [] arg);
//Evaluate
[DllImport("CxTranslator.dll",CallingConvention=CallingConvention.StdCall,
                        CharSet=CharSet.Ansi)]
        public extern static Complex Evaluate(string arg, string exp, Complex []
argVal);

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

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

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


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


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

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


        /// ////////////////////////////////////////////////////////////////
        //Unicode impl of CxTranslator
public class CxTranslatorW
{
//EvaluateEqu
[DllImport("CxTranslatorW.dll",CallingConvention=CallingConvention.StdCall,
                        CharSet=CharSet.Unicode)]
        public extern static  Complex EvaluateEqu(string exp, Complex [] arg);
//Evaluate
[DllImport("CxTranslatorW.dll",CallingConvention=CallingConvention.StdCall,
                        CharSet=CharSet.Unicode)]
        public extern static Complex Evaluate(string arg, string exp, Complex []
argVal);

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

//Delegate, AddFunction & RemoveFunction
        public delegate  Complex  CallBackUserFunction(ref Complex val);
//AddFunction &Remove Function
[DllImport("CxTranslatorW.dll",CallingConvention=CallingConvention.StdCall,
                        CharSet=CharSet.Unicode)]
        public extern static bool AddFunction(string
funName,CallBackUserFunction fn);
[DllImport("CxTranslatorW.dll",CallingConvention=CallingConvention.StdCall,
                        CharSet=CharSet.Unicode)]
        public extern static bool RemoveFunction(string exp);

//AddConst & RemoveConst
[DllImport("CxTranslatorW.dll",CallingConvention=CallingConvention.StdCall,
                        CharSet=CharSet.Unicode)]
        public extern static bool AddConst(string exp, Complex val);
[DllImport("CxTranslatorW.dll",CallingConvention=CallingConvention.StdCall,
                              CharSet=CharSet.Unicode)]
          public extern static bool RemoveConst(string exp);


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


public static void SetCxTranslatorException()
{
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 SetCxTranslatorException from CxTranslator(W) class. Example how to use CxTranslator exception in C# :

```
// Demo.cs :
//CxTranslator algorithm version 1.1
//Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
//http://www.e-MathSolution.com
//For details contact us: info@e-MathSolution.com

using System;

namespace CSharpTest
{
class Class1
{

//User defined function
          static EMS.Complex qub(ref EMS.Complex x)
          {
                    EMS.Complex res;
                    res.re=x.re*x.re*x.re;
                    res.im=x.im*x.im*x.im;
                    return res;
          }


[STAThread]
static void Main(string[] args)
{
          EMS.Complex [] arg= new EMS.Complex[2];
          EMS.Complex res=new EMS.Complex();
          EMS.Complex myConst= new EMS.Complex();

          string mExp="f(x,y)=x+(5i+y)^2-exp(x)/qub(x/y)+myConst";
          arg[0]=new EMS.Complex();
          arg[1]=new EMS.Complex();

          arg[0].re=5; arg[0].im=2;
          arg[1].re=2; arg[1].im=1;

          myConst.re=5.7;  myConst.im=2.4;

          //Delegate for the user defined function
          EMS.CxTranslator.CallBackUserFunction qubDelegate=
                    new EMS.CxTranslator.CallBackUserFunction(qub);

try
{
          EMS.CxTranslator.SetCxTranslatorException();//Set error handle
          EMS.CxTranslator.AddConst("myConst", myConst);
          EMS.CxTranslator.AddFunction("qub",qubDelegate);

          EMS.CxTranslator.BuildEqu("f(x,y)=sin(x)^2+x/y+6i+qub(myConst)");
          res=EMS.CxTranslator.CalcFor(arg);
          Console.WriteLine("res.re={0}   res.im={1}",res.re, res.im);

          res=EMS.CxTranslator.EvaluateEqu(mExp, arg);
          Console.WriteLine("res.re={0}   res.im={1}",res.re, res.im);

          EMS.CxTranslator.RemoveConst("myConst");
          res=EMS.CxTranslator.Evaluate("x,y", "(x+cos(y))-myConst", arg);
          Console.WriteLine("res.re={0}   res.im={1}",res.re, res.im);
}
catch(Exception e)
{
          Console.WriteLine(e.Message);
}
Console.ReadLine();
}
}
}
```

VB.NET

CxTranslator.vb should be included in a VB.NET project. The file contains EMS namespace with CxTranlator and CxTranlatorW class:

```
'CxTranslator.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

  <StructLayoutAttribute(LayoutKind.Sequential)> _
  Public Structure Complex
    Public re As Double
    Public im As Double
  End Structure

  Public Class CxTranslator
 <DllImport("CxTranslator.dll", CallingConvention:=CallingConvention.StdCall, _
   CharSet:=CharSet.Ansi)> _
    Public Shared Function EvaluateEqu(ByVal exp As String, ByRef args As
Complex) As Complex
    End Function
<DllImport("CxTranslator.dll", CallingConvention:=CallingConvention.StdCall, _
      CharSet:=CharSet.Ansi)> _
      Public Shared Function Evaluate(ByVal arg As String, ByVal exp As String,
ByRef args As Complex) As Complex
    End Function
<DllImport("CxTranslator.dll", CallingConvention:=CallingConvention.StdCall, _
      CharSet:=CharSet.Ansi)> _
      Public Shared Sub BuildEqu(ByVal arg As String)
    End Sub
<DllImport("CxTranslator.dll", CallingConvention:=CallingConvention.StdCall, _
      CharSet:=CharSet.Ansi)> _
      Public Shared Sub Build(ByVal arg As String, ByVal exp As String)
    End Sub
<DllImport("CxTranslator.dll", CallingConvention:=CallingConvention.StdCall, _
      CharSet:=CharSet.Ansi)> _
     Public Shared Function CalcFor(ByRef args As Complex) As Complex
    End Function

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

<DllImport("CxTranslator.dll", CallingConvention:=CallingConvention.StdCall, _
      CharSet:=CharSet.Ansi)> _
      Public Shared Function AddConst(ByVal constName As String, ByVal val
As Complex) As Boolean
    End Function
 <DllImport("CxTranslator.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("CxTranslator.dll", CallingConvention:=CallingConvention.StdCall, _
      CharSet:=CharSet.Ansi)> _
      Public Shared Sub SetErrorHandle(ByVal errDgt As CallBackDelegate)
    End Sub


    Public Sub SetCxTranslatorException()
      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 CxTranslator algorithm
  Public Class CxTranslatorW
 <DllImport("CxTranslatorW.dll", CallingConvention:=CallingConvention.StdCall, _
         CharSet:=CharSet.Unicode)> _
        Public Shared Function EvaluateEqu(ByVal exp As String, ByRef args
As Complex) As Complex
    End Function
<DllImport("CxTranslatorW.dll", CallingConvention:=CallingConvention.StdCall, _
      CharSet:=CharSet.Unicode)> _
        Public Shared Function Evaluate(ByVal arg As String, ByVal exp As
String, ByRef args As Complex) As Complex
    End Function
<DllImport("CxTranslatorW.dll", CallingConvention:=CallingConvention.StdCall, _
      CharSet:=CharSet.Unicode)> _
        Public Shared Sub BuildEqu(ByVal arg As String)
    End Sub
<DllImport("CxTranslatorW.dll", CallingConvention:=CallingConvention.StdCall, _
      CharSet:=CharSet.Unicode)> _
      Public Shared Sub Build(ByVal arg As String, ByVal exp As String)
    End Sub
<DllImport("CxTranslatorW.dll", CallingConvention:=CallingConvention.StdCall, _
      CharSet:=CharSet.Unicode)> _
     Public Shared Function CalcFor(ByRef args As Complex) As Complex
    End Function


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

<DllImport("CxTranslatorW.dll", CallingConvention:=CallingConvention.StdCall, _
      CharSet:=CharSet.Unicode)> _
      Public Shared Function AddConst(ByVal constName As String, ByVal val
As Complex) As Boolean
    End Function
<DllImport("CxTranslatorW.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("CxTranslatorW.dll", CallingConvention:=CallingConvention.StdCall, _
      CharSet:=CharSet.Unicode)> _
      Public Shared Sub SetErrorHandle(ByVal errDgt As CallBackDelegate)
    End Sub


    Public Sub SetCxTranslatorException()
      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 `SetCxTranslatorException` from `CxTranslator(W)` class. Example how to use CxTranslator exception in VB.NET :

```
' Demo.vb :
'CxTranslator 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() As EMS.Complex
      Dim res As EMS.Complex
      Dim tr As EMS.CxTranslator
      Dim pi2 As EMS.Complex
      Dim qb As EMS.CxTranslator.CallBackUserFunction

      arg = New EMS.Complex(3) {}
      tr = New EMS.CxTranslator()
      arg(0).re = 3
      arg(0).im = 2

      arg(1).re = 5.6
      arg(1).im = -5

      arg(2).re = 15.6
      arg(2).im = 3.12

      pi2.re = 6.28
      pi2.im = 0

      expr = "f(x,y,z)=sin(x)+x*(-y)+5.4789i*exp(sin(x/z))-pi2"

      qb = New EMS.CxTranslator.CallBackUserFunction(AddressOf qub)
      tr.AddFunction("qub", qb)
      tr.AddConst("pi2", pi2)

      tr.SetCxTranslatorException()

      res = tr.EvaluateEqu(expr, arg(0))
      System.Console.WriteLine("res.re={0}   res.im={1}", res.re, res.im)

      res = tr.Evaluate("x,y", "qub(x)+exp(y)", arg(0))
      System.Console.WriteLine("res.re={0}   res.im={1}", res.re, res.im)

      tr.RemoveConst("pi2")
      tr.BuildEqu(expr)
      res = tr.CalcFor(arg(0))
      System.Console.WriteLine("res.re={0}   res.im={1}", res.re, res.im)

errHnd:  'Error
      Dim s As String
      s = Err.Description
      System.Console.WriteLine(Err.Description)

   End Sub

   'User defined function
   Function qub(ByRef x As EMS.Complex) As EMS.Complex
      Dim res As EMS.Complex
      res.re = x.re * x.re * x.re
      res.im = x.im * x.im * x.im
      qub = res
   End Function
End Module
```

Demo of the algorithm

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

Demo.exe

The Demo for *CxTranslator* algorithm is presented as console application demo.exe, which is built up on CxTr.lib.PRC version and it gives you main information how to use *CxTranslator* 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 :
//CxTranslator 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>
#include <windows.h>

#include"CxTranslator.h"


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


cout<<"CxTranslator. The Demo Application."<<endl;
cout<<"Evaluate a run-time defined complex number math expressions."<<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<<"To designate image part of a complex number please use 'i' or 'j'
suffix."<<endl;
cout<<"Example:  F(x,y,z)=sin(x)-3+5i+y^z*sin(x/1.3j)"<<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<<"].real=";   cin>>re;
                cout<<"args["<<j+1<<"].image=";   cin>>im;
                args[j]=TCalc(re,im);
        }

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

cout<<"Please, wait..."<<endl;
InitCxTranslator();

try{
        GetLocalTime(&sysTimeStart);
        start = clock();

        BuildEqu(funcExp);
        for(long i=0; i<iterNum; i++)
                res+=CalcFor(args);

        finish = clock();
        GetLocalTime(&sysTimeEnd);
cout<<endl<<"=============================================="<<endl;
cout<<"Start time: "<<sysTimeStart.wMinute<<"(min)
"<<sysTimeStart.wSecond<<"(s) ";
cout<<sysTimeStart.wMilliseconds<<"(ms)"<<endl;

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;

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

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

CxTranslator.dll

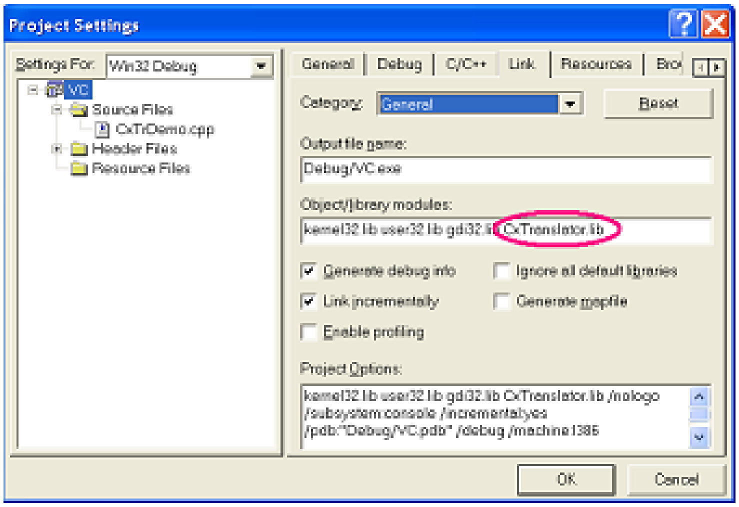
Also, *CxTranslator.zip* includes a demo version of the algorithm as *CxTranslator.dll* component and some project examples in VC++, C++Builder, VB, Delphi, C#, and VB.NET.
The *CxTranslator.dll* must be in active folder of the calling application to run properly. And, the demo version has been limited 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: Samples

Visual C++

CxTr.lib.PRC

CxTr.lib.PRC is very easy to use. Programmer must include head file #include "CxTranslator.h" and add CxTranslator.lib or CxTranslatorW.lib as appropriate in Project Setting window (see figure below). The functions InitCxTranslator() and CloseCxTranslator() must be called explicitly.



Settings For: Win32 Debug

Category: General

Output file name: Debug/VC.exe

Object/library modules: kernel32.lib user32.lib gdi32.lib CxTranslator.lib

☑ Generate debug info ☐ Ignore all default libraries

☑ Link incrementally ☐ Generate mapfile

☐ Enable profiling

Project Options:

kernel32.lib user32.lib gdi32.lib CxTranslator.lib /nologo
/subsystem:console /incremental:yes
/pdb:"Debug/VC.pdb" /debug /machine:I386

```
// Test.cpp :
//CxTranslator 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"CxTranslator.h"


const TCalc __stdcall toRad(const TCalc& x)
{
        return TCalc(x.real()*3.14/180,0);
}

int main(int argc, char* argv[])
{
     TCalc args[]= {TCalc(3.6,6),TCalc(5.6,-4),
                   TCalc(-1.5,0.33) };
    TCalc twoPi=TCalc(6.28,0);
    TCHAR *mathFunc = _T("F(x, y, z) = 3i*x^2-y^2+sin(toRad(45))/twoPi");
    TCHAR *mathExp = _T("3*x^2-y^2+cos(toRad(85))/4j");
    TCHAR *varList = _T("x y z");

   InitCxTranslator();

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

        CloseCxTranslator();
         return 0;
}
```

CxTr.lib.OOP

The Object Oriented version of *CxTranslator* 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 “CxTranslator.h”` and add CxTranslator.lib or CxTranslatorW.lib correspondingly to the Project Setting window (see CxTr.lib.PRC explanation above). CxTr.lib.OOP requires only `InitCxTranslator()` to be called.

```
// Test.cpp :
//CxTranslator 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"CxTranslator.h"


const TCalc __stdcall toRad(const TCalc& x)
{
                return TCalc(x.real()*3.14/180,0);
}

TCalc PassObject(CxTranslator &p)
{
                TCalc args[]={TCalc(0.6,1),TCalc(5.2,-1.4), TCalc(-7.5,123)};
                return p.CalcFor(x);
}

int main(int argc, char* argv[])
{
          TCalc args[]= {TCalc(3.6,6),TCalc(5.6,-4),
         TCalc(-1.5,0.33) };
        TCalc twoPi=TCalc(6.28,0);
        TCalc res;
        TCHAR *mathFunc = _T("F(x, y, z) = 3i*x^2-y^2+sin(toRad(45))/twoPi");
        TCHAR *mathExp = _T("3*x^2-y^2+cos(toRad(85))/4j");
        TCHAR *varList = _T("x y z");
                CxTranslator tr1, tr2;

                CxTranslator::InitCxTranslator();

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

                return 0;
}
```

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

CxTr.dll.PRC

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

```
// Test.cpp :
//CxTranslator 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"CxTranslator.h"


const TCalc __stdcall toRad(const TCalc& x)
{
                return TCalc(x.real()*3.14/180,0);
}

int main(int argc, char* argv[])
{
            TCalc args[] = {TCalc(3.6,6),TCalc(5.6,-4),
                                TCalc(-1.5,0.33) };
        TCalc res,twoPi= TCalc(6.28,0);
        TCHAR *mathFunc = _T("F(x, y, z) = 3i*x^2-y^2+sin(toRad(45))/twoPi");
        TSHAR *mathExp = _T("3*x^2-y^2+cos(toRad(85))/4j");
        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(CxException &e) {
            cout<<e.what().c_str();
    }

     return 0;
}
```

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

```
// Test.cpp :
//CxTranslator 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"CxTranslatorEx.h"


const TCalc __stdcall toRad(const TCalc& x)
{
            return TCalc(x.real()*3.14/180,0);
}

int main(int argc, char* argv[])
{
     TCalc args[[]= {TCalc(3.6,6),TCalc(5.6,-4),
TCalc(-1.5,0.33) };
     TCalc res,twoPi= TCalc(6.28,0);
     TCAHR *mathFunc = _T("F(x, y, z) = 3i*x^2-y^2+sin(toRad(45))/twoPi");
     TCHAR *mathExp = _T("3*x^2-y^2+cos(toRad(85))/4j");
     TCHAR *varList = _T("x y z");

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

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

     return 0;
}
```

CxTr.dll.OOP

Programmer should include head file #include "CxTranslator.h" and add CxTranslator.lib or CxTranslatorW.lib as appropriate in Project Setting window (see CxTr.lib.PRC explanation above). Iinitialization is not required.

```
// Test.cpp :
//CxTranslator 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"CxTranslator.h"


const TCalc __stdcall toRad(const TCalc& x)
{
            return TCalc(x.real()*3.14/180,0);
}

TCalc PassObject(CxTranslator &p)
{
            TCalc args[]={TCalc(0.6,1),TCalc(5.2,-1.4), TCalc(-7.5,123)};
            return p.CalcFor(x);
}

int main(int argc, char* argv[])
{
TCalc args[[]= {TCalc(3.6,6),TCalc(5.6,-4),
TCalc(-1.5,0.33) };
     TCalc res,twoPi= TCalc(6.28,0);
TCHAR *mathFunc = _T("F(x, y, z) = 3i*x^2-y^2+sin(toRad(45))/twoPi");
TCHAR *mathExp = _T("3*x^2-y^2+cos(toRad(85))/4j");
TCHAR *varList = _T("x y z");
            CxTranslator tr1, tr2;

            CxTranslator.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(CxException &e) {
        cout<<e.what().c_str();
}

return 0;
}
```

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

C++Builder

CxTr.dll.PRC

The same as for VC++, explicit call.

Delphi

CxTr.dll.PRC

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

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

  windows,
  CxTranslator in 'CxTranslator.pas';

function toRad(var x: TComplex):TComplex; stdcall;
var res:TComplex;
begin
    res.Im:=0;
    res.re:=x.re*3.14/180;
    toRad:=res;
end;

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

begin

args[0].Re:=3.6;args[0].Im:=6;
args[1].Re:=5.6;args[1].Im:=-4;
args[2].Re:=-1.5;args[2].Im:=0.33;
twoPi.Re:=6.28;    twoPi.Im:=0;

 mathFunc := 'f(x,y,z)=sin(x)+x*(-y)+5.4789i*exp(sin(x/(toRad(y))))-cos(z+3)/(y-
                          x^z)+twoPi';
 mathExp := '3*x^2-y^2+cos(85)/4i+twoPi';
 varList := 'x y z';

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

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

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

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

         RemoveConst('twoPi');
         res :=Evaluate(varList, mathExp, @args);
         writeln('Re=',res.Re,'   Im=',res.Im);


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

     end;

  readln;
end
```

VB

CxTr.dll.PRC

To use the component in VB, just add CxTranslator.bas to VB project. The source code of CxTranslator.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()
On Error GoTo errHnd
   Dim arg(3) As Complex
    Dim res As Complex


   On Error GoTo errHnd
   arg(0).re = 3.6
   arg(0).im = 6
   arg(1).re = 5.6
   arg(1).im = -4
     arg(2).re = -1.5
     arg(2).im = 0.33

   expr$ = "f(x,y,z)=sin(x)+qub(x*(-y))+5.4789i*exp(sin(x/(y)))-cos(z+3)"

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

   BuildEqu expr$
   res = CalcFor(arg(0))
   Label1.Caption = Str(res.re)
   Label2.Caption = Str(res.im)

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

   Exit Sub

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

C#

CxTr.dll.PRC

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

```
// Demo.cs :
//CxTranslator algorithm version 1.1
//Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
//http://www.e-MathSolution.com
//For details contact us: info@e-MathSolution.com

using System;

namespace CSharpTest
{
class Class1
{

//User defined function
        static EMS.Complex qub(ref EMS.Complex x)
        {
                EMS.Complex res;
                res.re=x.re*x.re*x.re;
                res.im=x.im*x.im*x.im;
                return res;
        }


[STAThread]
static void Main(string[] args)
{
        EMS.Complex [] arg= new EMS.Complex[2];
        EMS.Complex res=new EMS.Complex();
        EMS.Complex myConst= new EMS.Complex();

        string mExp="f(x,y)=x+(5i+y)^2-exp(x)/qub(x/y)+myConst";
        arg[0]=new EMS.Complex();
        arg[1]=new EMS.Complex();

        arg[0].re=5; arg[0].im=2;
        arg[1].re=2; arg[1].im=1;

        myConst.re=5.7;  myConst.im=2.4;

        //Delegate for the user defined function
        EMS.CxTranslator.CallBackUserFunction qubDelegate=
                new EMS.CxTranslator.CallBackUserFunction(qub);

try
{
        EMS.CxTranslator.SetCxTranslatorException();//Set error handle
        EMS.CxTranslator.AddConst("myConst", myConst);
        EMS.CxTranslator.AddFunction("qub",qubDelegate);

        EMS.CxTranslator.BuildEqu("f(x,y)=sin(x)^2+x/y+6i+qub(myConst)");
        res=EMS.CxTranslator.CalcFor(arg);
        Console.WriteLine("res.re={0}   res.im={1}",res.re, res.im);

        res=EMS.CxTranslator.EvaluateEqu(mExp, arg);
        Console.WriteLine("res.re={0}   res.im={1}",res.re, res.im);

        EMS.CxTranslator.RemoveConst("myConst");
        res=EMS.CxTranslator.Evaluate("x,y", "(x+cos(y))-myConst", arg);
        Console.WriteLine("res.re={0}   res.im={1}",res.re, res.im);
}
catch(Exception e)
{
        Console.WriteLine(e.Message);
}
Console.ReadLine();
}
}
}
```

VB.NET

CxTr.dll.PRC

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

```
Demo.vb :
'CxTranslator 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() As EMS.Complex
      Dim res As EMS.Complex
      Dim tr As EMS.CxTranslator
      Dim pi2 As EMS.Complex
      Dim qb As EMS.CxTranslator.CallBackUserFunction

      arg = New EMS.Complex(3) {}
      tr = New EMS.CxTranslator()
      arg(0).re = 3
      arg(0).im = 2

      arg(1).re = 5.6
      arg(1).im = -5
```

```
        arg(2).re = 15.6
        arg(2).im = 3.12

        pi2.re = 6.28
        pi2.im = 0

        expr = "f(x,y,z)=sin(x)+x*(-y)+5.4789i*exp(sin(x/z))-pi2"

        qb = New EMS.CxTranslator.CallBackUserFunction(AddressOf qub)
        tr.AddFunction("qub", qb)
        tr.AddConst("pi2", pi2)

        tr.SetCxTranslatorException()

        res = tr.EvaluateEqu(expr, arg(0))
        System.Console.WriteLine("res.re={0}   res.im={1}", res.re, res.im)

        res = tr.Evaluate("x,y", "qub(x)+exp(y)", arg(0))
        System.Console.WriteLine("res.re={0}   res.im={1}", res.re, res.im)

        tr.RemoveConst("pi2")
        tr.BuildEqu(expr)
        res = tr.CalcFor(arg(0))
        System.Console.WriteLine("res.re={0}   res.im={1}", res.re, res.im)

errHnd:  'Error
        Dim s As String
        s = Err.Description
        System.Console.WriteLine(Err.Description)

    End Sub

    'User defined function
    Function qub(ByRef x As EMS.Complex) As EMS.Complex
        Dim res As EMS.Complex
        res.re = x.re * x.re * x.re
        res.im = x.im * x.im * x.im
        qub = res
    End Function
End Module
```