AcidLib

Manual

Blue Analysis
AI Technology Simplified

Preface

The Acid Library is a formula parser developed by Blue Analysis. It was designed for speed and efficiency and therefore performs an outstanding job at large mathematical operation. The Acid Parser is ideal for any large mathematical task, and supports plenty of mathematical functions, ranging from basic mathematics to trigonometric functions and complex numbers. The Math Parser also supports evaluation signs and custom functions. It is purely based on the Microsoft .Net Framework, making it both more stable and more secure than any ordinary Win32 application. This also provides the advantage of easy linking and fast performance in other .Net Applications and support for all the latest Microsoft .Net Frameworks.

Contents

Features

Real Numbers

The Acid Library supports all real numbers and a collection of standard trigonometric and special functions. See each function for more details.

Imaginary & Complex Numbers

The Acid Library supports imaginary and real numbers as well. Imaginary numbers are defined with either the letter “i” or the letter “j”, and can afterwards be used as pleased. See each function for more details.

User-Defined Functions

The Acid Library supports custom user-defined functions utilizing both real and complex numbers. User-defined functions may call each other, and the possibilities are therefore endless. The only requirement for the user-defined functions is that their names don’t overlap with other constants or functions in the library.

Microsoft .Net Framework Support

The Acid Library supports all the latest Microsoft .Net Frameworks:

- Microsoft .Net Compact Framework 3.5
- Microsoft .Net Compact Framework 2.0
- Microsoft .Net Framework 3.5
- Microsoft .Net Framework 3.0
- Microsoft .Net Framework 2.0

Functions

Overview

The Acid Parser consists of several functions with different purposes. Generally the functions can be split up into 3 groups: standard math parsing functions, math parsing functions with user-defined equations and evaluation functions.

Main parsing functions for math expressions:

- ParseSimpleReal
    - Addition, Subtraction, Multiplication, Division & Brackets
- ParseAdvancedReal
    - Addition, Subtraction, Multiplication, Division & Brackets
    - Trigonometric Functions & Special Functions
- ParseSimpleComplex
    - Addition, Subtraction, Multiplication, Division & Brackets
    - Imaginary & Complex Numbers
- ParseAdvancedComplex
    - Addition, Subtraction, Multiplication, Division & Brackets
    - Trigonometric Functions & Special Functions
    - Imaginary & Complex Numbers

Math parsing functions for custom user-defined equations:

- ParseCustomReal
    - Addition, Subtraction, Multiplication, Division & Brackets
    - Trigonometric Functions & Special Functions
    - User-defined equations
- ParseCustomComplex
    - Addition, Subtraction, Multiplication, Division & Brackets
    - Trigonometric Functions & Special Functions
    - Imaginary & Complex Numbers
    - User-defined equations

Evaluation functions:

- EvaluateSimpleReal
    - Evaluation signs: =, !=, <, >, <=, =>
    - Addition, Subtraction, Multiplication, Division & Brackets
- EvaluateAdvancedReal
    - Evaluation signs: =, !=, <, >, <=, =>
    - Addition, Subtraction, Multiplication, Division & Brackets
    - Trigonometric Functions & Special Functions
- EvaluateSimpleComplex
    - Evaluation signs: =, !=, <, >, <=, =>
    - Addition, Subtraction, Multiplication, Division & Brackets
    - Imaginary & Complex Numbers
- EvaluateAdvancedComplex
    - Evaluation signs: =, !=, <, >, <=, =>
    - Addition, Subtraction, Multiplication, Division & Brackets
    - Trigonometric Functions & Special Functions
    - Imaginary & Complex Numbers
- EvaluateCustomReal
    - Evaluation signs: =, !=, <, >, <=, =>
    - Addition, Subtraction, Multiplication, Division & Brackets
    - Trigonometric Functions & Special Functions
    - User-defined equations
- EvaluateCustomComplex
    - Evaluation signs: =, !=, <, >, <=, =>
    - Addition, Subtraction, Multiplication, Division & Brackets
    - Trigonometric Functions & Special Functions
    - Imaginary & Complex Numbers
    - User-defined equations

ParseSimpleReal

ParseSimpleReal is the simple parsing function.. It supports brackets () and performs the basic mathematical expressions:

- Addition: +
- Subtraction: -
- Multiplication: *
- Division: /
- Power: ^

You should use this when high performance is required, and the input expression doesn't contain anything but the basic mathematical operators and parentheses. C# Example:

```
using Acid_Library;

...

string formula, result;

AcidParser MyParser = new AcidParser("User", "Serial");

formula = "8-(2+2)*(20-2)";

result = MyParser.ParseSimpleReal(formula);
```

ParseAdvancedReal

ParseAdvancedReal is the advanced parsing function. Supports brackets, performs basic mathematical expressions, but also calculates trigonometric and special functions:

- Addition: +
- Subtraction: -
- Multiplication: *
- Division: /
- Power: ^

- Absolute Value: abs(x)
- Cosine: cos(x)
- Sinus: sin(x)
- Tan: tan(x)
- (Radians)Inverse Cosine: acos(x)
- (Radians)Inverse Sinus: asin(x)
- (Radians)Inverse Tan: atan(x)
- (Radians)Hyperbolic Cosine: cosh(x)
- (Radians)Hyperbolic Sinus: sinh(x)
- (Radians)Hyperbolic Tan: tanh(x)
- Logarithm to e: log(x)
- Logarithm to 10: log10(x)
- Degrees to Radians function: rad(x)
- Radians to Degrees function: deg(x)
- Constant Pi: pi
- Constant E: ee

While ParseSimpleReal is faster than ParseAdvancedReal, ParseAdvancedReal is still recommended in most cases because of its additional functions. C# Example:

```
using Acid_Library;

...

string formula, result;

AcidParser MyParser = new AcidParser("User", "Serial");

formula = "sin(10)*cos(10) / pi";

result = MyParser.ParseAdvancedReal(formula);
```

ParseSimpleComplex

ParseSimpleComplex is the simple parsing function.. It supports brackets () and performs the basic mathematical expressions on real, imaginary and complex numbers. :

- Addition: +
- Subtraction: -
- Multiplication: *
- Division: /
- Power: ^

Imaginary numbers are denoted by either the letter "i" or the letter "j". You should use this when high performance is required, and the input expression doesn't contain anything but the basic mathematical operators and parentheses as well as real and complex numbers. C# Example:

```
using Acid_Library;

...

string formula, result;

AcidParser MyParser = new AcidParser("User", "Serial");

formula = "i8-(i2+2)*(20-i2)";

result = MyParser.ParseSimpleComplex(formula);
```

ParseAdvancedComplex

ParseAdvancedComplex is the advanced parsing function. Supports brackets, performs basic mathematical expressions on real, imaginary and complex numbers. Supports trigonometric and special functions as well:

- Addition: +
- Subtraction: -
- Multiplication: *
- Division: /
- Power: ^

- Absolute Value: abs(x)
- Cosine: cos(x)
- Sinus: sin(x)
- Tan: tan(x)
- (Radians)Inverse Cosine: acos(x)
- (Radians)Inverse Sinus: asin(x)
- (Radians)Inverse Tan: atan(x)
- (Radians)Hyperbolic Cosine: cosh(x)
- (Radians)Hyperbolic Sinus: sinh(x)
- (Radians)Hyperbolic Tan: tanh(x)
- Logarithm to e: log(x)
- Logarithm to 10: log10(x)
- Degrees to Radians function: rad(x)
- Radians to Degrees function: deg(x)
- Constant Pi: pi
- Constant E: ee

- Real: re(x)
- Imaginary: im(x)
- Modulus: mod(x)
- Conjugate: conj(x)

Imaginary numbers are denoted by either the letter “i” or the letter “j”. While ParseSimpleComplex is faster than ParseAdvancedComplex, ParseAdvancedComplex is still recommended in most cases because of its additional functions. C# Example:

```
using Acid_Library;

...

string formula, result;

AcidParser MyParser = new AcidParser("User", "Serial");

formula = "sin(2+i3)*cos(i3+5) / pi";

result = MyParser.ParseAdvancedComplex(formula);
```

ParseCustomReal

ParseCustomReal is the advanced parsing function for custom functions and real numbers. Supports brackets, performs basic mathematical expressions, but also calculates trigonometric, special functions and user-defined functions:

- Addition: +
- Subtraction: -
- Multiplication: *
- Division: /
- Power: ^

- Absolute Value: abs(x)
- Cosine: cos(x)
- Sinus: sin(x)
- Tan: tan(x)
- (Radians)Inverse Cosine: acos(x)
- (Radians)Inverse Sinus: asin(x)
- (Radians)Inverse Tan: atan(x)
- (Radians)Hyperbolic Cosine: cosh(x)
- (Radians)Hyperbolic Sinus: sinh(x)
- (Radians)Hyperbolic Tan: tanh(x)
- Logarithm to e: log(x)
- Logarithm to 10: log10(x)
- Degrees to Radians function: rad(x)
- Radians to Degrees function: deg(x)
- Constant Pi: pi
- Constant E: ee

Creating a user-defined function is easy. Using it afterwards only requires one to use semi-colons between the parameters. C# Example:

NB: Variables may be confused with functions. Therefore it is recommended to use underscores.

```
using Acid_Library;

...

string formula, result;

string nfunction = "wow";

string [] pfunction = new string[2]{"_x", "_y"};

string efunction = "(_x^2+_y^2)^0.5";


AcidParser MyParser = new AcidParser("User", "Serial");

Pars.AddCustomFunction(nfunction, pfunction, efunction)


formula = "wow(51;32)";

result = MyParser.ParseCustomReal(formula);
```

ParseCustomComplex

ParseCustomComplex is the advanced parsing function for custom functions and complex numbers. Supports brackets, performs basic mathematical expressions and calculates trigonometric and special functions on real, imaginary and complex numbers with user-defined functions:

- Addition: +
- Subtraction: -
- Multiplication: *
- Division: /
- Power: ^

- Absolute Value: abs(x)
- Cosine: cos(x)
- Sinus: sin(x)
- Tan: tan(x)
- (Radians)Inverse Cosine: acos(x)
- (Radians)Inverse Sinus: asin(x)
- (Radians)Inverse Tan: atan(x)
- (Radians)Hyperbolic Cosine: cosh(x)
- (Radians)Hyperbolic Sinus: sinh(x)
- (Radians)Hyperbolic Tan: tanh(x)
- Logarithm to e: log(x)
- Logarithm to 10: log10(x)
- Degrees to Radians function: rad(x)
- Radians to Degrees function: deg(x)
- Constant Pi: pi
- Constant E: ee

- Real: re(x)
- Imaginary: im(x)
- Modulus: mod(x)
- Conjugate: conj(x)

Creating a user-defined function is easy. Using it afterwards only requires one to use semi-colons between the parameters. Complex numbers may naturally be used as well. C# Example:

NB: Variables may be confused with functions. Therefore it is recommended to use underscores.

```
using Acid_Library;

...

string formula, result;

string nfunction = "wow";

string [] pfunction = new string[2]{"x", "y"};

string efunction = "(re(x)^2+im(y)^2)^(2+i0.5)";


AcidParser MyParser = new AcidParser("User", "Serial");

Pars.AddCustomFunction(nfunction, pfunction, efunction)


formula = "wow(51;32)";

result = MyParser.ParseCustomReal(formula);
```

Evaluation Functions

All the evaluation functions support the following evaluation operators:

- Equal sign: =
- Unequal sign: !=
- Less than: <
- More than: >
- Less or equal to: <=
- More or equal to: =>

All evaluation functions will either return true or false. One can choose to use individual evaluation functions if high performance is required or simply use the EvaluateCustomComplex, which supports all functions including user-defined equations. All evaluation functions support extended equations, such as: 24 + 32 > 31 + 4 = 33+2 < 100. C# Example with EvaluateCustomComplex: NB: Variables may be confused with functions. Therefore it is recommended to use underscores.

```
using Acid_Library;

...

string formula, result;

string nfunction = "wow";

string [] pfunction = new string[2]{"x", "y"};

string efunction = "(re(x)^2+im(y)^2)^(2+i0.5)";


AcidParser MyParser = new AcidParser("User", "Serial");

Pars.AddCustomFunction(nfunction, pfunction, efunction)


formula = "wow(51;32) < wow(31;42) => wow(42;31)";

result = MyParser.EvaluateCustomReal(formula);
```

Performance Benchmark

The Acid Library is one of the fastest math parsers available to date. Most math parsers use slow string manipulation methods, but the Acid Library takes advantage of a genuine design and minimal string manipulation, increasing its performance 100 times compared to regular math parsers.

Computer Specifications:

- Microsoft Windows XP
- AMD Athlon™ 64 X2 Dual Core Processor 5200+ 2,70 GHz
- NVIDIA GeForce 7300 GS
- 240 GB HDD
- 2 GB Ram

Microsoft .Net Framework 3.5

ParseSimpleReal	**(Microsoft .Net Framework 3.5)**		
Equation	10000 Iterations	100000 Iterations	1000*000* Iterations (One Million)
3+6	0.036s	0.323s	3.276s
5-3	0.031s	0.328s	3.276s
2*4	0.036s	0.323s	3.245s
7/3	0.036s	0.37s	3.667s
3^2	0.031s	0.333s	3.365s
1+2+3+4	0.062s	0.656s	6.594s
1-2-3-4	0.073s	0.75s	7.469s
1*2*3*4	0.068s	0.667s	6.641s
1/2/3/4	0.083s	0.839s	8.37s
1^2^3^4	0.068s	0.667s	6.641s

ParseAdvancedReal	**(Microsoft .Net Framework 3.5)**		
Equation	10000 Iterations	100000 Iterations	1000000 Iterations (One Million)
abs(-3)	0.089s	0.885s	8.693s
log(3)	0.125s	1.26s	12.531s
log10(3)	0.13s	1.302s	12.87s
cos(3)	0.13s	1.292s	12.927s
sin(3)	0.125s	1.271s	12.76s
asin(0.3)	0.13s	1.328s	13.302s
tan(3)	0.13s	1.318s	13.229s
atan(0.3)	0.135s	1.339s	13.25s
rad(3)	0.125s	1.286s	12.927s
deg(3)	0.125s	1.26s	12.464s

ParseSimpleComplex	**(Microsoft .Net Framework 3.5)**		
Equation	10000 Iterations	100000 Iterations	1000000 Iterations (One Million)
(3+i6)+(4+i3)	0.161s	1.589s	15.557s
(3+i6)-(3+i2)	0.156s	1.578s	15.682s
(2+i3)*(4+i2)	0.156s	1.589s	15.729s
(2+i3)/(4+i2)	0.161s	1.604s	15.964s
(2+i3)^(4+i2)	0.182s	1.797s	18.021s
(3+i6)+(4+i3)+(2+i6)+(1+i7)	0.333s	3.339s	33.432s
(3+i6)-(4+i3)-(2+i6)-(1+i7)	0.349s	3.495s	34.891s
(3+i6)*(4+i3)*(2+i6)*(1+i7)	0.354s	3.578s	35.656s
(3+i6)/(4+i3)/(2+i6)/(1+i7)	0.365s	3.661s	36.396s
(3+i6)^(4+i3)^(2+i6)^(1+i7)	0.432s	4.302s	43.016s

ParseAdvancedComplex	**(Microsoft .Net Framework 3.5)**		
Equation	10000 Iterations	100000 Iterations	1000000 Iterations (One Million)
abs(-3+i2)	0.188s	1.807s	18.078s
log(3+i2)	0.224s	2.198s	21.995s
log10(3+i2)	0.229s	2.266s	22.703s
cos(3+i2)	0.24s	2.365s	23.729s
sin(3+i2)	0.219s	2.182s	21.818s
asin(0.3+i2)	0.25s	2.531s	25.318s
tan(3)	0.125s	1.26s	12.583s
atan(0.3+i2)	0.24s	2.411s	24.109s
conj(0.3+i2)	0.125s	1.25s	12.453s
arg(0.3+i2)	0.125s	1.24s	12.411s

Microsoft .Net Framework 3.0

ParseSimpleReal	**(Microsoft .Net Framework 3.0)**		
Equation	10000 Iterations	100000 Iterations	1000000 Iterations (One Million)
3+6	0.031s	0.328s	3.276s
5-3	0.036s	0.328s	3.297s
2*4	0.031s	0.323s	3.26s
7/3	0.036s	0.365s	3.672s
3^2	0.031s	0.333s	3.359s
1+2+3+4	0.062s	0.661s	6.583s
1-2-3-4	0.073s	0.75s	7.448s
1*2*3*4	0.068s	0.667s	6.531s
1/2/3/4	0.083s	0.833s	8.312s
1^2^3^4	0.068s	0.667s	6.635s

ParseAdvancedReal	**(Microsoft .Net Framework 3.0)**		
Equation	10000 Iterations	100000 Iterations	1000000 Iterations (One Million)
abs(-3)	0.094s	0.885s	8.792s
log(3)	0.12s	1.245s	12.5s
log10(3)	0.125s	1.286s	12.859s
cos(3)	0.13s	1.292s	12.911s
sin(3)	0.125s	1.276s	12.62s
asin(0.3)	0.13s	1.323s	13.25s
tan(3)	0.135s	1.323s	13.229s
atan(0.3)	0.135s	1.333s	13.151s
rad(3)	0.13s	1.276s	12.885s
deg(3)	0.125s	1.24s	12.401s

ParseSimpleComplex	**(Microsoft .Net Framework 3.0)**		
Equation	10000 Iterations	100000 Iterations	1000000 Iterations (One Million)
(3+i6)+(4+i3)	0.161s	1.589s	15.453s
(3+i6)-(3+i2)	0.156s	1.552s	15.568s
(2+i3)*(4+i2)	0.156s	1.583s	15.755s
(2+i3)/(4+i2)	0.161s	1.635s	16s
(2+i3)^(4+i2)	0.182s	1.807s	17.974s
(3+i6)+(4+i3)+(2+i6)+(1+i7)	0.333s	3.349s	33.411s
(3+i6)-(4+i3)-(2+i6)-(1+i7)	0.349s	3.49s	35.01s
(3+i6)*(4+i3)*(2+i6)*(1+i7)	0.359s	3.568s	35.646s
(3+i6)/(4+i3)/(2+i6)/(1+i7)	0.365s	3.661s	36.406s
(3+i6)^(4+i3)^(2+i6)^(1+i7)	0.432s	4.292s	42.906s

ParseAdvancedComplex	**(Microsoft .Net Framework 3.0)**		
Equation	10000 Iterations	100000 Iterations	1000000 Iterations (One Million)
abs(-3+i2)	0.188s	1.812s	18.078s
log(3+i2)	0.219s	2.208s	22.021s
log10(3+i2)	0.229s	2.271s	22.714s
cos(3+i2)	0.234s	2.385s	23.87s
sin(3+i2)	0.219s	2.188s	21.849s
asin(0.3+i2)	0.255s	2.542s	25.302s
tan(3)	0.125s	1.26s	12.609s
atan(0.3+i2)	0.24s	2.417s	24.115s
conj(0.3+i2)	0.125s	1.25s	12.49s
arg(0.3+i2)	0.125s	1.245s	12.432s

Microsoft .Net Framework 2.0

ParseSimpleReal	**(Microsoft .Net Framework 2.0)**		
Equation	10000 Iterations	100000 Iterations	1000000 Iterations (One Million)
3+6	0.036s	0.328s	3.281s
5-3	0.036s	0.323s	3.286s
2*4	0.031s	0.328s	3.266s
7/3	0.036s	0.37s	3.693s
3^2	0.031s	0.328s	3.312s
1+2+3+4	0.068s	0.651s	6.562s
1-2-3-4	0.073s	0.75s	7.464s
1*2*3*4	0.062s	0.667s	6.635s
1/2/3/4	0.083s	0.833s	8.318s
1^2^3^4	0.062s	0.667s	6.635s

ParseAdvancedReal	**(Microsoft .Net Framework 2.0)**		
Equation	10000 Iterations	100000 Iterations	1000000 Iterations (One Million)
abs(-3)	0.094s	0.891s	8.844s
log(3)	0.125s	1.255s	12.552s
log10(3)	0.13s	1.286s	12.896s
cos(3)	0.13s	1.318s	13.286s
sin(3)	0.13s	1.286s	12.729s
asin(0.3)	0.13s	1.339s	13.422s
tan(3)	0.13s	1.312s	13.255s
atan(0.3)	0.135s	1.333s	13.234s
rad(3)	0.13s	1.297s	12.922s
deg(3)	0.125s	1.255s	12.464s

ParseSimpleComplex	**(Microsoft .Net Framework 2.0)**		
Equation	10000 Iterations	100000 Iterations	1000000 Iterations
(3+i6)+(4+i3)	0.161s	1.594s	15.589s
(3+i6)-(3+i2)	0.156s	1.573s	15.536s
(2+i3)*(4+i2)	0.161s	1.589s	15.755s
(2+i3)/(4+i2)	0.161s	1.63s	16.151s
(2+i3)^(4+i2)	0.182s	1.812s	18.062s
(3+i6)+(4+i3)+(2+i6)+(1+i7)	0.333s	3.344s	33.438s
(3+i6)-(4+i3)-(2+i6)-(1+i7)	0.354s	3.495s	34.948s
(3+i6)*(4+i3)*(2+i6)*(1+i7)	0.359s	3.573s	35.76s
(3+i6)/(4+i3)/(2+i6)/(1+i7)	0.365s	3.661s	36.495s
(3+i6)^(4+i3)^(2+i6)^(1+i7)	0.432s	4.307s	43.01s

ParseAdvancedComplex	(Microsoft .Net Framework 2.0)		
Equation	10000 Iterations	100000 Iterations	1000000 Iterations
abs(-3+i2)	0.188s	1.812s	18.078s
log(3+i2)	0.219s	2.214s	22.068s
log10(3+i2)	0.229s	2.276s	22.75s
cos(3+i2)	0.24s	2.37s	23.74s
sin(3+i2)	0.219s	2.182s	21.833s
asin(0.3+i2)	0.25s	2.536s	25.167s
tan(3)	0.125s	1.266s	12.661s
atan(0.3+i2)	0.245s	2.422s	24.208s
conj(0.3+i2)	0.125s	1.245s	12.453s
arg(0.3+i2)	0.125s	1.24s	12.391s

Benchmark C# Program Code

```
using System;
using Acid_Library;

namespace Atest
{
 class Program
  {
  public static void Main(string[] args)
   {
   string U = "Username";
   string P = ”Serial";

   AcidParser Pars = new AcidParser(U, P);

   int [] IterationCount = new int[3];
   IterationCount[0] = 10000;
   IterationCount[1] = 100000;
   IterationCount[2] = 1000000;

   string [] Simp = new string[10];
   Simp[0] = "3+6";
   Simp[1] = "5-3";
   Simp[2] = "2*4";
   Simp[3] = "7/3";
   Simp[4] = "3^2";
   Simp[5] = "1+2+3+4";
   Simp[6] = "1-2-3-4";
   Simp[7] = "1*2*3*4";
   Simp[8] = "1/2/3/4";
   Simp[9] = "1^2^3^4";

   string [] Advan = new string[10];
   Advan[0] = "abs(-3)";
   Advan[1] = "log(3)";
   Advan[2] = "log10(3)";
   Advan[3] = "cos(3)";
   Advan[4] = "sin(3)";
   Advan[5] = "asin(0.3)";
   Advan[6] = "tan(3)";
   Advan[7] = "atan(0.3)";
   Advan[8] = "rad(3)";
   Advan[9] = "deg(3)";
```

```
string [] SimpCom = new string[10];
SimpCom[0] = "(3+i6)+(4+i3)";
SimpCom[1] = "(3+i6)-(3+i2)";
SimpCom[2] = "(2+i3)*(4+i2)";
SimpCom[3] = "(2+i3)/(4+i2)";
SimpCom[4] = "(2+i3)^(4+i2)";
SimpCom[5] = "(3+i6)+(4+i3)+(2+i6)+(1+i7)";
SimpCom[6] = "(3+i6)-(4+i3)-(2+i6)-(1+i7)";
SimpCom[7] = "(3+i6)*(4+i3)*(2+i6)*(1+i7)";
SimpCom[8] = "(3+i6)/(4+i3)/(2+i6)/(1+i7)";
SimpCom[9] = "(3+i6)^(4+i3)^(2+i6)^(1+i7)";

string [] AdvanCom = new string[10];
AdvanCom[0] = "abs(-3+i2)";
AdvanCom[1] = "log(3+i2)";
AdvanCom[2] = "log10(3+i2)";
AdvanCom[3] = "cos(3+i2)";
AdvanCom[4] = "sin(3+i2)";
AdvanCom[5] = "asin(0.3+i2)";
AdvanCom[6] = "tan(3)";
AdvanCom[7] = "atan(0.3+i2)";
AdvanCom[8] = "conj(0.3+i2)";
AdvanCom[9] = "arg(0.3+i2)";

string Tmp = "";

string OutputFile = "ParseSimpleReal" + Environment.NewLine + "Equation, ";
for (int i = 0; i < 3; i++)
{
 OutputFile += ", " + IterationCount[i].ToString() + " Iterations";
}

OutputFile += Environment.NewLine;
for (int n = 0; n < 10; n++)
{
 OutputFile += Simp[n];
 for (int i = 0; i < 3; i++)
 {
  double SecondCount = 0;
  for (int j = 0; j < 3; j++)
  {
   DateTime StartTime = DateTime.Now;
   for (int k = 0; k < IterationCount[i]; k++)
   {
    Tmp = Pars.ParseSimpleReal(Simp[n]);
   }
   DateTime EndTime = DateTime.Now;
   SecondCount += EndTime.Subtract(StartTime).TotalSeconds;
  }

  Console.WriteLine(Simp[n] + "=" + Tmp);

  SecondCount = SecondCount / 3;
  OutputFile += ", " + Math.Round(SecondCount, 3).ToString() + "s";
 }
 OutputFile += Environment.NewLine;
}

OutputFile += Environment.NewLine + "ParseAdvancedReal" + Environment.NewLine + "Equation";
for (int i = 0; i < 3; i++)
{
 OutputFile += ", " + IterationCount[i].ToString() + " Iterations";
}
```

```
OutputFile += Environment.NewLine;
for (int n = 0; n < 10; n++)
{
 OutputFile += Advan[n];
 for (int i = 0; i < 3; i++)
 {
  double SecondCount = 0;
  for (int j = 0; j < 3; j++)
  {
   DateTime StartTime = DateTime.Now;
   for (int k = 0; k < IterationCount[i]; k++)
   {
    Tmp = Pars.ParseAdvancedReal(Advan[n]);
   }
   DateTime EndTime = DateTime.Now;
   SecondCount += EndTime.Subtract(StartTime).TotalSeconds;
  }

  Console.WriteLine(Advan[n] + "=" + Tmp);

  SecondCount = SecondCount / 3;
  OutputFile += ", " + Math.Round(SecondCount, 3).ToString() + "s";
 }
 OutputFile += Environment.NewLine;
}

OutputFile += Environment.NewLine + "ParseSimpleComplex" + Environment.NewLine + "Equation";
for (int i = 0; i < 3; i++)
{
 OutputFile += " " + IterationCount[i].ToString() + "Iterations";
}

OutputFile += Environment.NewLine;
for (int n = 0; n < 10; n++)
{
 OutputFile += SimpCom[n];
 for (int i = 0; i < 3; i++)
 {
  double SecondCount = 0;
  for (int j = 0; j < 3; j++)
  {
   DateTime StartTime = DateTime.Now;
   for (int k = 0; k < IterationCount[i]; k++)
   {
    Tmp = Pars.ParseSimpleComplex(SimpCom[n]);
   }
   DateTime EndTime = DateTime.Now;
   SecondCount += EndTime.Subtract(StartTime).TotalSeconds;
  }
  Console.WriteLine(SimpCom[n] + "=" + Tmp);

  SecondCount = SecondCount / 3;
  OutputFile += " " + Math.Round(SecondCount, 3).ToString() + "s";
 }
 OutputFile += Environment.NewLine;
}

OutputFile += Environment.NewLine + "ParseAdvancedComplex" + Environment.NewLine + "Equation";
for (int i = 0; i < 3; i++)
{
 OutputFile += " " + IterationCount[i].ToString() + " Iterations";
}
```

```
   OutputFile += Environment.NewLine;
   for (int n = 0; n < 10; n++)
   {
    OutputFile += AdvanCom[n];
    for (int i = 0; i < 3; i++)
    {
     double SecondCount = 0;
     for (int j = 0; j < 3; j++)
     {
      DateTime StartTime = DateTime.Now;
      for (int k = 0; k < IterationCount[i]; k++)
      {
       Tmp = Pars.ParseAdvancedComplex(AdvanCom[n]);
      }
      DateTime EndTime = DateTime.Now;
      SecondCount += EndTime.Subtract(StartTime).TotalSeconds;
     }

     Console.WriteLine(AdvanCom[n] + "=" + Tmp);

     SecondCount = SecondCount / 3;
     OutputFile += " " + Math.Round(SecondCount, 3).ToString() + "s";
    }
    OutputFile += Environment.NewLine;
   }

   Console.Write("Writing output to file...");
   System.IO.FileStream fs = new System.IO.FileStream("c:\\Benchmark.txt", System.IO.FileMode.OpenOrCreate,
System.IO.FileAccess.Write, System.IO.FileShare.ReadWrite);
   System.IO.StreamWriter sw = new System.IO.StreamWriter(fs, System.Text.Encoding.ASCII);
   sw.Write(OutputFile);
   sw.Flush();
   sw.Close();
   fs.Close();

   Console.Write("Press any key to continue . . . ");
   Console.ReadKey(true);
  }
 }
}
```

Error Handling

Developers and end-users can easily commit typing or syntax mistakes in their expressions. Therefore the Acid Library provides a general exception handling making it easy to identify the invalid syntax or function call. Following is the list of returnable errors:

#1: Error: Calculating the square of a negative number. Irrational numbers not supported.
This error is usually triggered by expressions taking the root of a negative number, example: (-5)^0.5

#2: Error: Division by zero is an illegal mathematical operation.
This error is usually triggered by expressions dividing by zero, example: 34/0

#3: Error: Tried calculating a non-numeric value 'x'.
This error is usually triggered by expressions containing non-numeric values, such as x+y

#4: Error: Tried calculating a non-numeric value using an advanced function 'x'.
This error is usually triggered by expressions containing non-numeric values inside advanced functions, such as cos(x+y)

#5: Error: Missing or illegal combination of brackets.
This error is usually triggered by an illegal combination of brackets.

#6: Error: Expression contains white spaces.
This error is only triggered by the existence of white spaces in the expressions, example: 10 +2.

#7: Error: Missing or illegal combination of mathematical operators.
This error is usually triggered by an illegal or missing combination of brackets or mathematical operators.

#8: Error: Illegal use of mathematical operators or brackets.
This error is usually triggered by an illegal or unequal number of brackets or mathematical operators.

#9: Error: Illegal use of mathematical functions on complex or imaginary numbers.
This error is usually triggered by using a function, meant only for real numbers, on a complex number. Illegal functions include: rad() and deg().

#10: Error: Function 'x' doesn't exist.
This error is usually triggered by a function call, to a custom function which doesn’t exist.

#11: Error: Wrong function parameters for function 'x' with illegal parameters 'y'.
This error is usually triggered by a function call with illegal parameters.

#12: Error: Unknown Error.
Please notice that this error will not be seen as “unknown” error but rather as a system exception, which will describe the error in depth.