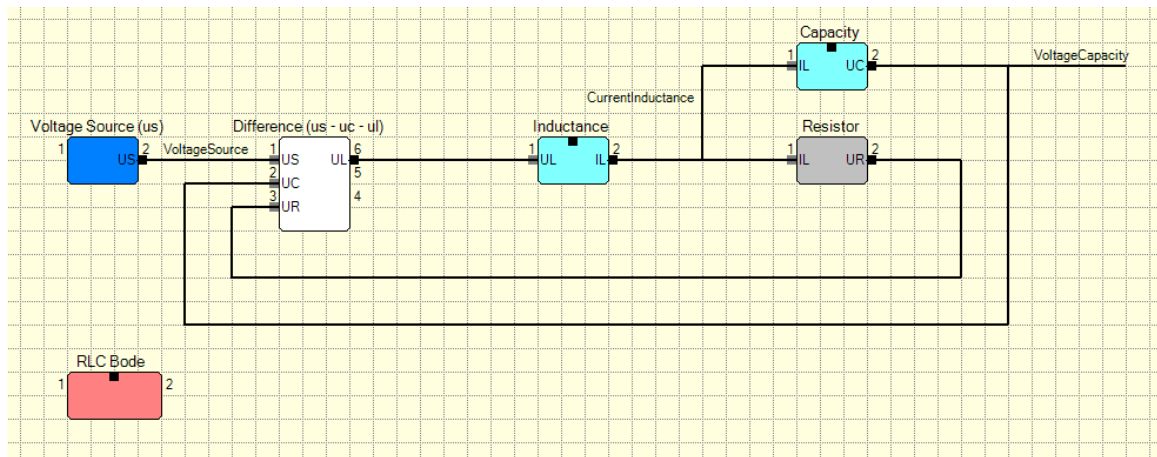


Brace*Calculation*

Studio 5



User Reference



Manufacturer Information



Eicher Engineering
Software Development
Janahofstrasse 12
CH-8722 Kaltbrunn SG
Switzerland

www.eichereng.ch
info@eichereng.ch

© 2001-2026 by Eicher Engineering



Table of Contents

1	Before You Begin.....	9
1.1	General.....	9
1.2	Installation, Uninstallation	9
1.3	Manufacturer Information.....	9
1.3.1	Trademarks	9
1.3.2	Copyright.....	9
1.4	Licensing Agreement	10
2	Getting Started	13
2.1	Introduction	13
2.2	System Time Function $y=f(t)$	14
2.3	User Function $y=f(x)$	18
3	How BraceCalculation Works.....	21
3.1	Overview.....	21
3.2	Signal Array	22
3.3	Calculation	23
3.4	Function and Signal Behavior.....	24
3.5	Controlling Tracks	25
4	Using BraceCalculation.....	26
4.1	Introduction	26
4.2	Create a new Environment	26
4.3	Create a new System.....	26
4.3.1	Create Drawing Areas	27
4.3.2	Create Timing Frame	28
4.3.3	Create new Elements.....	29
4.3.4	Create new Pins	30
4.3.5	Create new Curves	31
4.3.6	Draw Wires	33
4.4	Write your Element Code	34
4.5	Build and Compile System	35
4.6	Calculate and Simulate System	36
4.7	System Control with Tracks	38
4.8	Write an Assembly	39
5	Reference.....	40
5.1	Start and Exit	40
5.2	File Handling.....	41
5.3	About BraceCalculation	42
5.4	Help	43



5.5	Calculation Controls	44
5.6	Additional Controls	45
5.7	Outputs	46
5.8	Output Events.....	48
5.9	Output Build	48
5.10	Schematic Area.....	49
5.11	Schematic Drawing.....	51
5.12	Settings	52
5.12.1	General.....	52
5.12.2	Engine Timing.....	53
5.12.3	Engine Tracks	54
5.12.3.1	Edit Track	55
5.12.4	Areas.....	56
5.12.4.1	Edit Area.....	57
5.12.5	Elements.....	58
5.12.5.1	Edit Element.....	59
5.12.5.2	Edit Pin	62
5.12.5.3	Options Elements	63
5.12.5.4	Options Pins	64
5.12.6	Curves.....	65
5.12.6.1	Edit Curve and Net	66
5.12.6.2	Edit Wire.....	67
5.12.6.3	Edit Axis	68
5.12.6.4	Edit Y Curves.....	70
5.12.6.5	Edit Y Curve	71
5.12.6.6	Options Scope	72
5.12.7	Text Fields	74
5.12.7.1	Edit Text Field	75
5.13	Element Dialog	77
5.13.1	Element Overview	78
5.13.2	Element User Code.....	79
5.13.3	Code Snippet.....	80
5.13.4	Element Scope.....	81
5.13.5	Element Local Event	83
5.13.6	Element Assembly.....	84
5.14	Engine Code Base	85
5.14.1	Calculation Interface	85
5.14.1.1	Init()	85
5.14.1.2	Loop()	85



5.14.1.3	Post()	85
5.14.2	Curve Functions	86
5.14.2.1	GetCurveName()	86
5.14.2.2	GetNumberCurves()	86
5.14.2.3	GetXSteps()	86
5.14.2.4	GetXDoubleStep()	86
5.14.2.5	GetXDoubleAbsBegin()	86
5.14.2.6	GetXDoubleActBegin()	87
5.14.2.7	GetXDoubleAbsEnd()	87
5.14.2.8	GetXDoubleActEnd()	87
5.14.2.9	GetXDouble()	87
5.14.2.10	GetXDoubleAbsIndex()	87
5.14.2.11	GetXDoubleActIndex()	88
5.14.2.12	GetXIntStep()	88
5.14.2.13	GetXIntAbsBegin()	88
5.14.2.14	GetXIntActBegin()	88
5.14.2.15	GetXIntAbsEnd()	88
5.14.2.16	GetXIntActEnd()	89
5.14.2.17	GetXInt()	89
5.14.2.18	GetXIntAbsIndex()	89
5.14.2.19	GetXIntActIndex()	89
5.14.2.20	GetYDoubleFromXDouble	90
5.14.2.21	GetYDoubleFromXInt()	90
5.14.2.22	GetYDouble()	90
5.14.2.23	GetYDoublePrev()	91
5.14.2.24	GetYDoubleAbsIndex()	92
5.14.2.25	GetYDoubleActIndex()	92
5.14.2.26	SetYDouble()	92
5.14.2.27	SetYDoubleAbsIndex()	93
5.14.2.28	GetYIntFromXDouble	94
5.14.2.29	GetYIntFromXInt()	94
5.14.2.30	GetYInt()	94
5.14.2.31	GetYIntPrev()	95
5.14.2.32	GetYIntAbsIndex()	96
5.14.2.33	GetYIntActIndex()	96
5.14.2.34	SetYInt()	96
5.14.2.35	SetYIntAbsIndex()	97
5.14.2.36	GetYByteArrayFromXDouble	98
5.14.2.37	GetYByteArrayFromXInt()	98



5.14.2.38	GetYByteArray()	98
5.14.2.39	GetYByteArrayPrev()	99
5.14.2.40	GetYByteArrayAbsIndex()	100
5.14.2.41	GetYByteArrayActIndex()	100
5.14.2.42	SetYByteArray()	100
5.14.2.43	SetYByteArrayAbsIndex()	101
5.15	Engine Library	102
5.15.1	Analog Logic	102
5.15.1.1	CreateAnalogLogic()	102
5.15.1.2	CreateAnalogStream()	102
5.15.1.3	CurveBUF()	103
5.15.1.4	CurveNOT()	104
5.15.1.5	CurveOR()	105
5.15.1.6	CurveNOR()	106
5.15.1.7	CurveXOR()	107
5.15.1.8	CurveAND()	108
5.15.1.9	CurveNAND()	109
5.15.1.10	CurveStream()	110
5.15.1.11	ActBuf()	111
5.15.1.12	ActNOT()	112
5.15.1.13	ActOR()	113
5.15.1.14	ActNOR()	114
5.15.1.15	ActXOR()	115
5.15.1.16	ActAND()	116
5.15.1.17	ActNAND()	117
5.15.1.18	ActStream()	118
5.15.2	Filter	119
5.15.2.1	CreateAnalogPolynom()	119
5.15.2.2	CreateFIR()	120
5.15.2.3	CreateIIR()	121
5.15.2.4	CurveFilterAmplitudeResponse()	122
5.15.2.5	CurveFilterPhaseResponse()	123
5.15.2.6	ActFiltering()	124
5.15.3	Operation	125
5.15.3.1	CreateIntegration()	125
5.15.3.2	CreateRMA()	125
5.15.3.3	CreateRMAInt()	125
5.15.3.4	CreateRMS()	125
5.15.3.5	CurveDerivative()	126



5.15.3.6	CurveIntegration()	127
5.15.3.7	CurveRMA()	128
5.15.3.8	CurveRMAInt()	129
5.15.3.9	CurveRMS()	130
5.15.3.10	CurveAdd()	131
5.15.3.11	CurveAddInt()	132
5.15.3.12	CurveSum()	133
5.15.3.13	CurveSumInt()	134
5.15.3.14	CurveDiff()	135
5.15.3.15	CurveDiffInt()	136
5.15.3.16	CurveMul()	137
5.15.3.17	CurveMulInt()	138
5.15.3.18	CurveProduct()	139
5.15.3.19	CurveProductInt()	140
5.15.3.20	CurveDiv()	141
5.15.3.21	CurveDivInt()	142
5.15.3.22	CurveFactorOffset()	143
5.15.3.23	CurveFactorOffsetInt()	144
5.15.3.24	CurveConvertDB()	145
5.15.3.25	CurveComposition()	146
5.15.3.26	ActDerive()	147
5.15.3.27	ActIntegrate()	148
5.15.3.28	ActRMA()	149
5.15.3.29	ActRMAInt()	150
5.15.3.30	ActRMS()	151
5.15.3.31	ActAdd()	152
5.15.3.32	ActAddInt()	153
5.15.3.33	ActSum()	154
5.15.3.34	ActSumInt()	155
5.15.3.35	ActDiff()	156
5.15.3.36	ActDiffInt()	157
5.15.3.37	ActMul()	158
5.15.3.38	ActMulInt()	159
5.15.3.39	ActProduct()	160
5.15.3.40	ActProductInt()	161
5.15.3.41	ActDiv()	162
5.15.3.42	ActDivInt()	163
5.15.3.43	ActFactorOffset()	164
5.15.3.44	ActFactorOffsetInt()	165



5.15.3.45	ActConvertDB()	166
5.15.3.46	ActComposition()	167
5.15.4	Signal	168
5.15.4.1	CurvePhase()	168
5.15.4.2	CurveMagnitude()	169
5.15.4.3	CurveIQBaseband()	170
5.15.4.4	ActPhase()	171
5.15.4.5	ActMagnitude()	172
5.15.4.6	ActIQBaseband()	173
5.15.5	Source	174
5.15.5.1	CreateSinus()	174
5.15.5.2	CreateLinear()	175
5.15.5.3	CreateRandom()	176
5.15.5.4	CurveSinus()	177
5.15.5.5	CurveLinear()	178
5.15.5.6	CurveRandom()	179
5.15.5.7	ActSinus()	180
5.15.5.8	ActLinear()	181
5.15.5.9	ActRandom()	182
5.15.6	Spectrum	183
5.15.6.1	CurveFourier()	183
5.15.6.2	CurveFFT()	184
5.16	Engine Track	185
5.16.1	Data Structure	185
5.16.2	Master Element	186
5.16.2.1	WriteGlobalEvent()	186
5.16.2.2	WriteLocalEvent()	186
5.16.2.3	Write()	186
5.16.2.4	Read()	186
5.16.3	Master Element to Assembly	187
5.16.3.1	Transmit()	187
5.16.3.2	Receive()	187
5.16.4	Slave Assembly Information	188
5.16.4.1	Subscribe()	188
5.16.4.2	WriteGlobalEvent()	188
5.16.4.3	WriteLocalEvent()	188
5.16.5	Slave Assembly to Element	189
5.16.5.1	Transmit()	189
5.16.5.2	Receive()	189



6	Examples	190
6.1	Series Oscillation Circuit.....	190
6.1.1	Overview	190
6.1.2	BraceCalculation Simple Implementation.....	191
6.1.3	BraceCalculation Simple Simulation	195
6.1.4	BraceCalculation Extended Implementation	197
6.1.5	BraceCalculation Extended Simulation	205
6.1.6	BraceCalculation Assembly Implementation	207
6.1.7	BraceCalculation Assembly DLL Creation	216
6.1.8	BraceCalculation Assembly Simulation	220



1 Before You Begin

1.1 General

BraceCalculation is independent offline software. It provides graphic presentations of mathematical calculations based on user C# code.

BraceCalculation runs as x64 application on Microsoft Windows 11.

Main software aspects:

- Windows program with state-of-the-art user interface.
- Drawing areas for signal visualization.
- Element objects for calculation definition.
- User defined C# code.
- User library.

1.2 Installation, Uninstallation

BraceCalculation is delivered as a Microsoft Installer format file MSI. It can be installed on every PC with Windows 11.

BraceCalculation can be uninstalled through the system control.

1.3 Manufacturer Information

1.3.1 Trademarks

Windows 11 is registered trademark of the Microsoft Corporation.

1.3.2 Copyright

It is not allowed to copy or change parts of this software. Please read the licensing agreement.



1.4 Licensing Agreement

The use of **BraceCalculation** is governed by the following terms and conditions:

Acceptance of License Agreement

You should carefully read the following terms and conditions before using **BraceCalculation** (the 'Software'). Unless you have a different license agreement signed by Eicher Engineering, your use of this software indicates your agreement to these terms and conditions.

If you do not accept all of these terms and conditions, you must cease using the Software immediately.

Copyright

Customer acknowledges that the Software, License Key and accompanying user documentation

('Documentation') are copyrighted works owned by Eicher Engineering and that Customer has no rights in the foregoing except as expressly granted herein.

Community Edition

Eicher Engineering hereby grants you an unlimited license to use the Software free of charge without a valid License Key.

The community edition contains following restrictions:

- Maximum numbers of Drawing Areas: 1
- Maximum numbers of Elements: 10
- Maximum numbers of Curve Indexes: 2

Professional Edition

The professional edition with a valid license key from Eicher Engineering grants you unlimited software access.

**License Key**

A unique key that will allow you to use the Software without any limited functions may be purchased from Eicher Engineering.

License Keys may be purchased for single or multiple users, and are priced according to the price list from Eicher Engineering.

Upon purchase of a License Key, Eicher Engineering hereby grants you a nonexclusive, non-transferable license to use the Software as follows:

Personal License Key:

A single-user License Key may either be used by a single user who uses the Software personally on one or more computers or installed on a single computer used by multiple people, but not both.

Multi-User License Key:

A multiple-user License Key may be purchased based upon either (a) the number of users who have access to the Software on any number of computers, or (b) based upon the number of computers on which the Software will be installed for use by any number of users. You may not use the Software in excess of either (a) the number of purchased users or (b) the number of purchased computers, whichever is applicable. A Multi-User License Key may not be used by your subsidiary companies, customers, or any other third parties.

You shall be responsible for maintaining the License Key in a safe location and are specifically prohibited from distributing the License Key, intentionally or unintentionally, to any third party. Upon loss or distribution of the License Key, Customer shall be required to pay a reinstatement fee at Eicher Engineering's discretion.

**Distribution of the Software**

Provided that you do not copy or distribute the License Key, and you include a copy of this License Agreement, you may (a) make copies of the Software; (b) give exact, unmodified copies of the Software to anyone; and (c) distribute the Software in its unmodified form via electronic means. You are specifically prohibited from charging any fees for any such copies or distributions.

Term and Termination

You may continue to use the Software for as long as you comply with the terms and conditions of this License Agreement. Eicher Engineering may terminate this License Agreement immediately upon notice to you in the event that Eicher Engineering has reason to believe you have breached this License Agreement. Upon termination, you shall immediately cease all use of the Software, License Key and Documentation and shall not be entitled to a refund of any fees paid.

Governing Law

The validity and interpretation of this Agreement shall be governed by the laws of Switzerland and the Kanton of St.Gallen. Customer agrees that the federal and state courts located in Switzerland and Kanton St.Gallen, shall be the appropriate site of venue for actions relating to this Agreement, and hereby consents to the exclusive jurisdiction and venue of such courts.

Disclaimer of Warranty

THE SOFTWARE AND DOCUMENTATION ARE PROVIDED 'AS IS.' TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, EICHER ENGINEERING DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE. ANY LIABILITY OF EICHER ENGINEERING WILL BE LIMITED EXCLUSIVELY TO REFUND OF THE PURCHASE PRICE



2 Getting Started

2.1 Introduction

In this chapter we don't go into deep details yet. Simple examples will be used to illustrate BraceCalculation basics.

With BraceCalculation you can define mathematical functions and process these functions with any operation.

The function behavior is given of your own C# code which is written inside BraceCalculation.

You can structure your functions and signals graphically using Elements and Areas.

An Element is a basic part of BraceCalculation.

It contains

- Pin definitions
- Curve definitions
- User C# code
- Calculation results and Scopes.

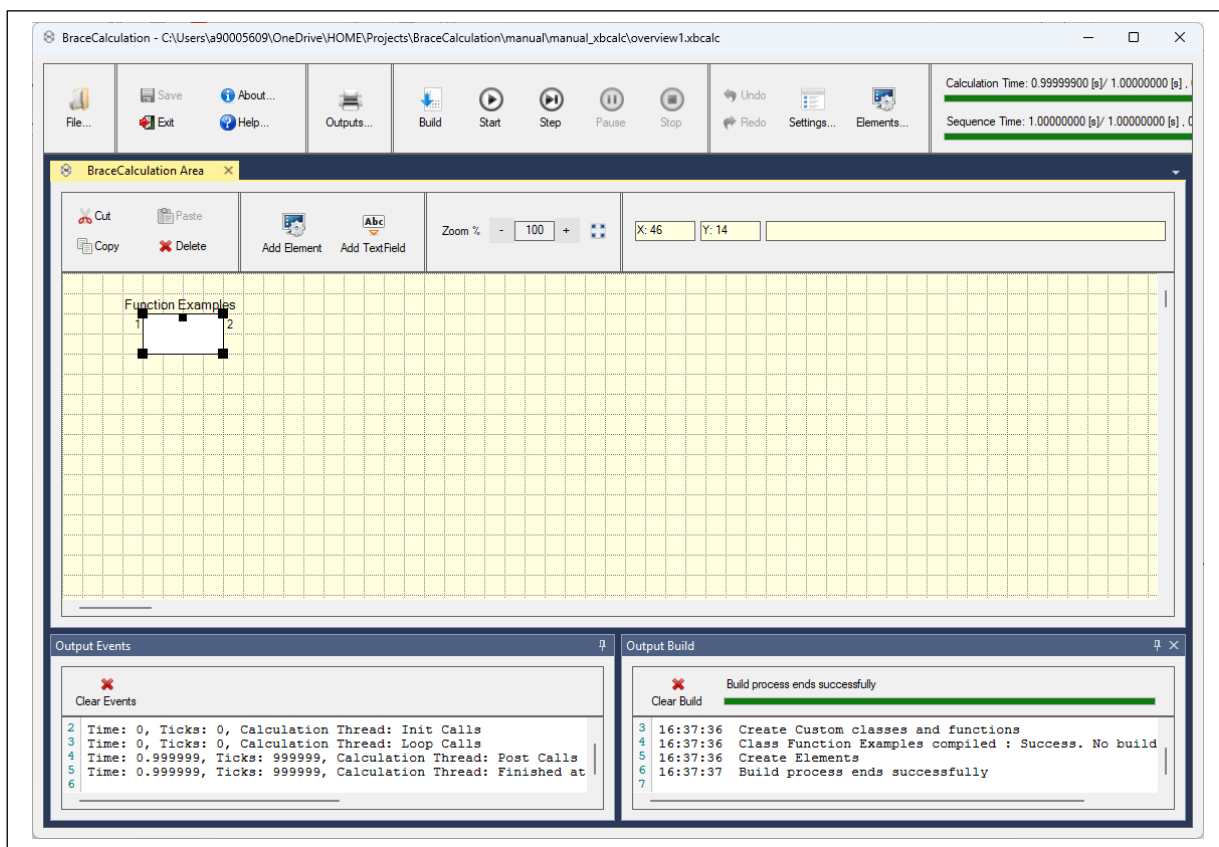


Figure 1 BraceCalculation Main Window

2.2 System Time Function $y=f(t)$

First, we want to calculate and plot a time-dependent sine curve.

$$y(t) = A \cdot \sin(2\pi \cdot f \cdot t + \varphi), 0 \leq t \leq t_{max}$$

A Amplitude
 f Frequency [Hz]
 φ Phase shift [rad]
 t Time [s] (BraceCalculation system time)
 t_{max} End system time

In BraceCalculation click <Add Element>

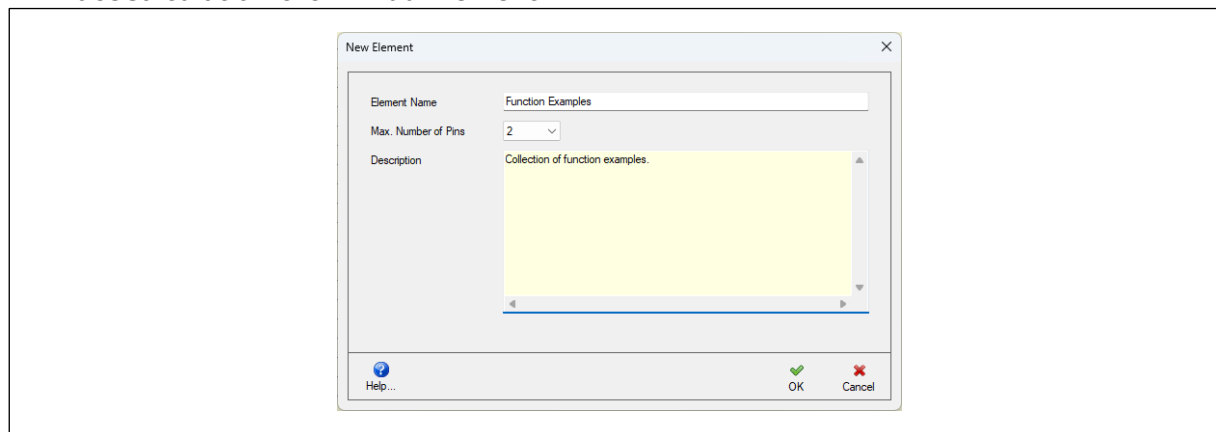


Figure 2 New Element Dialog

Right mouse button over the new Element and choose <Settings...>

- The Element setting dialog appears.
- Click on <Pins> Tab.

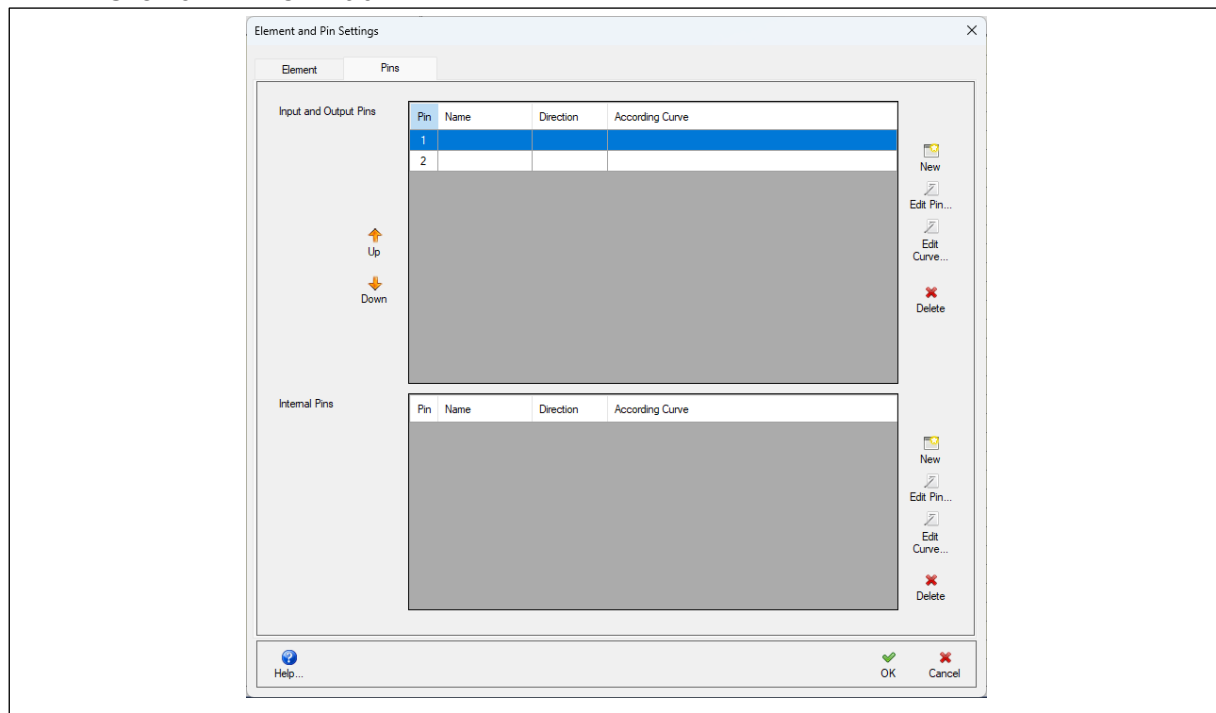


Figure 3 Element Settings Dialog

We use internal Pins for our functions.

Click <New> in the Internal Pins sections.

Click <Edit Curve...> to open the Curve dialog for the marked Pin.

Give a Curve name (no spaces) and a short description.

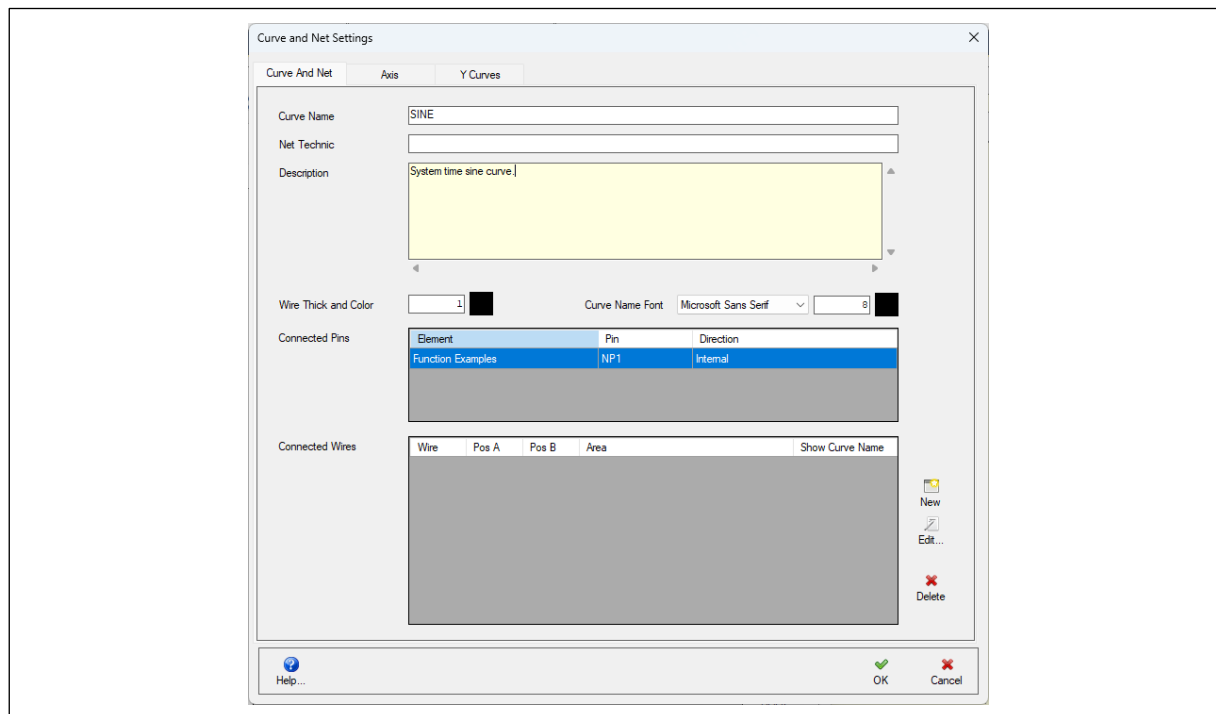


Figure 4 Curve Settings Dialog

Click <OK> and <OK>

Now our Element is defined with an internal Pin and a Curve 'SINE'.

It's time to define the functional behavior itself.

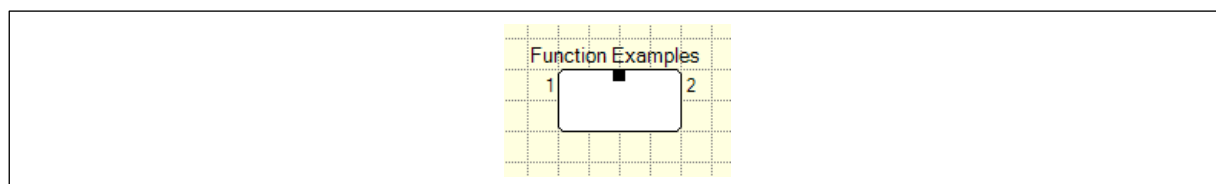


Figure 5 Element for our functions

Click left mouse button over Element and choose <Element...>

The Elements dialog appears. Here you have all access for every defined Element.

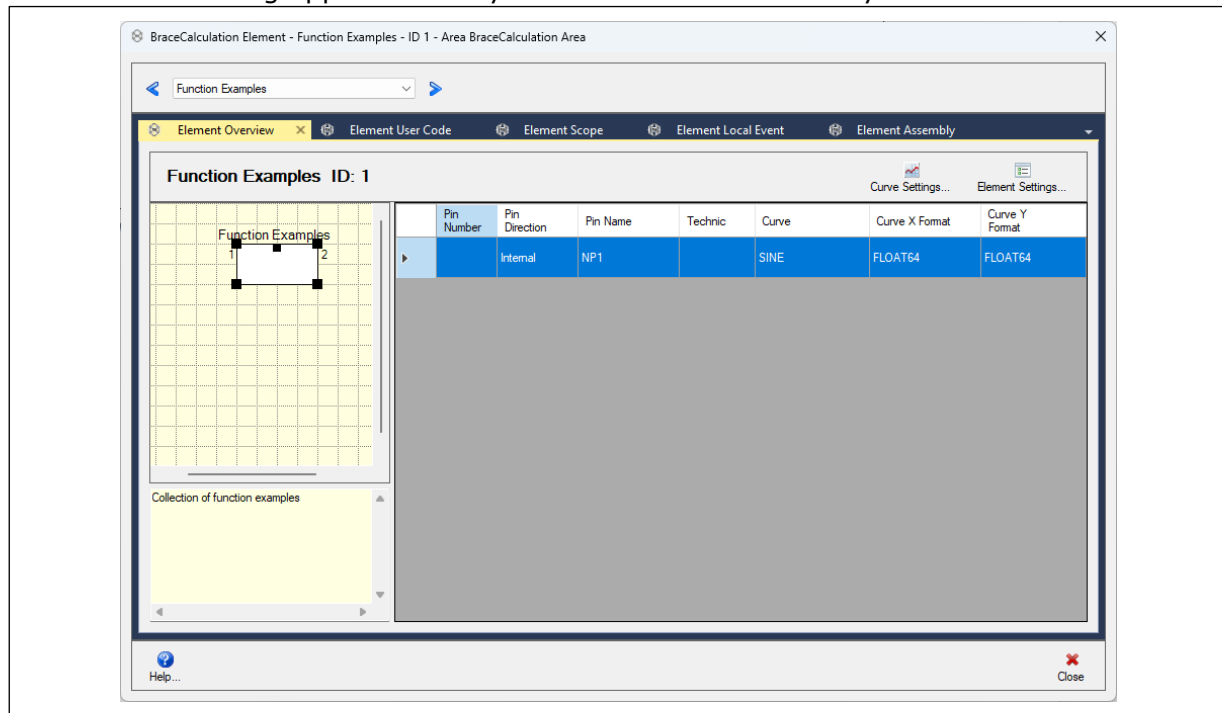


Figure 6 Elements Dialog

Now we define our sine behavior within the code window of the Element. Click on <Element User Code> and write the additional code below.

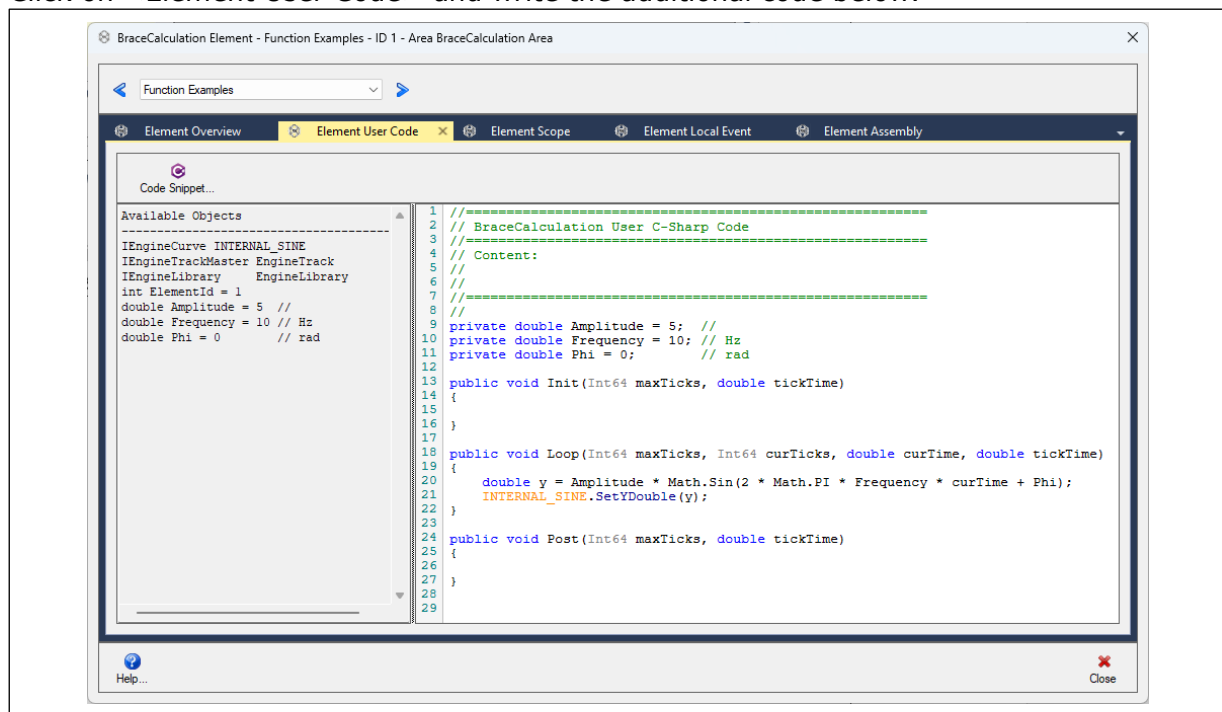


Figure 7 Elements User Code

Now everything is defined and coded.

In the BraceCalculation main window:

- Click <Save> to store the work.
- Click <Build> to compile the user code.
- Click <Start> to start the calculation.

Everything should be fine without any Errors in the build event window.

Click on <Element Scope> in the Element Dialog.

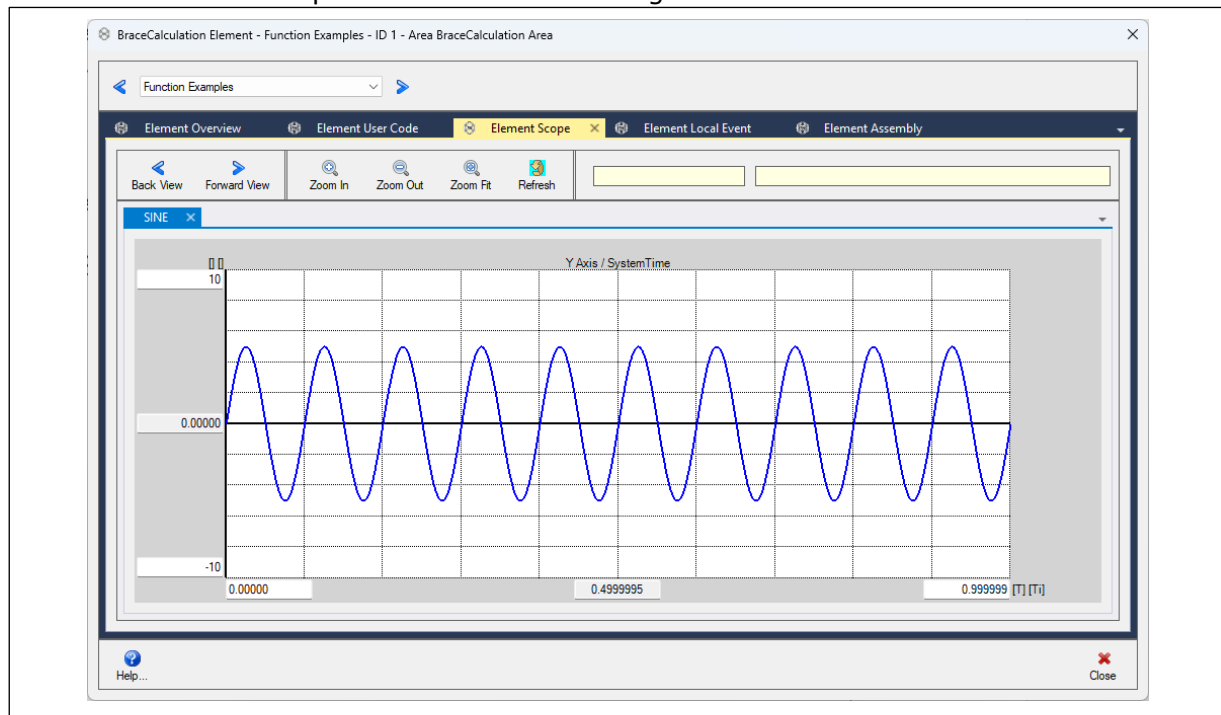


Figure 8 Elements SINE Scope



2.3 User Function $y=f(x)$

Now we can add a second Pin with a sine Curve to our Function Examples Element. But now, the curve $f(x)$ is user defined.

$$y(x) = A \cdot \sin(2\pi \cdot f \cdot x + \varphi), x_{\min} \leq x \leq x_{\max}$$

- A Amplitude
f Frequency [Hz]
 φ Phase shift [rad]
x x values within user range.
 $x_{\min}$ Begin user range
 $x_{\max}$ End user range

Click <New> in the Internal Pins sections.

Click <Edit Curve...> to open the Curve dialog for the marked Pin.

Give a Curve name (no spaces) and a short description.

Set <User> as Base instead <SystemTime> in the Axis Tab.

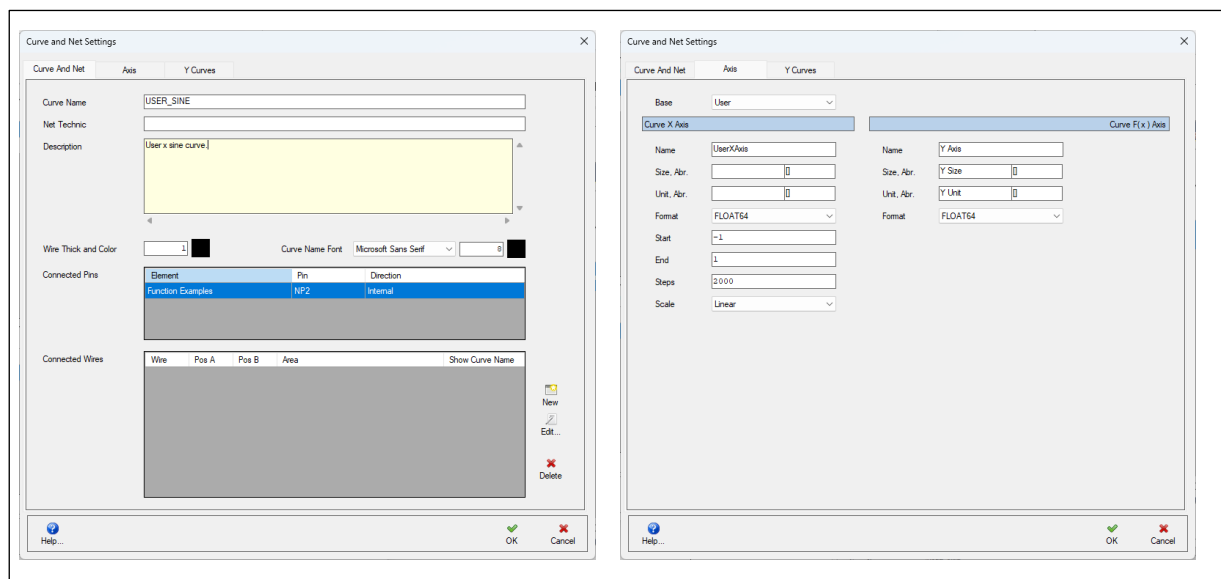


Figure 9 Curve and Axis Settings Dialog

Click <OK> and <OK>

Now we see both internal Pins and Curves in the Elements Dialog.

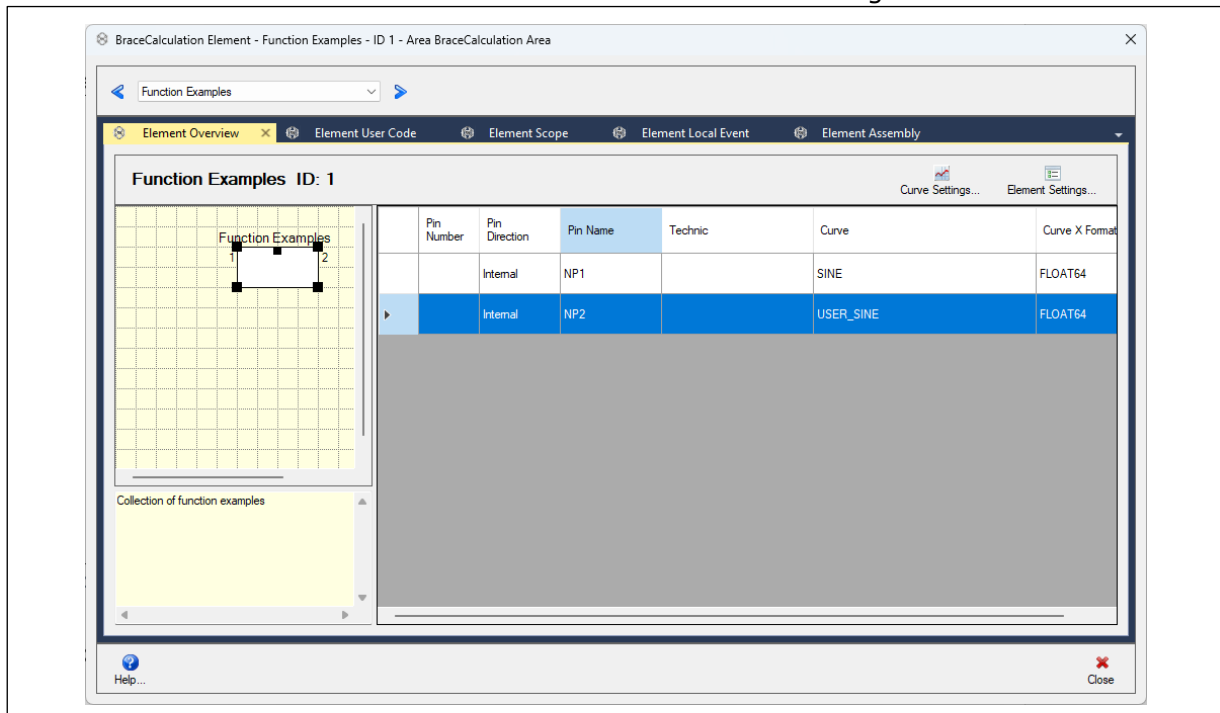


Figure 10 Elements Dialog

Now we define our user sine behavior within the code window of the Element. Click on <Element User Code> and write the additional code below.

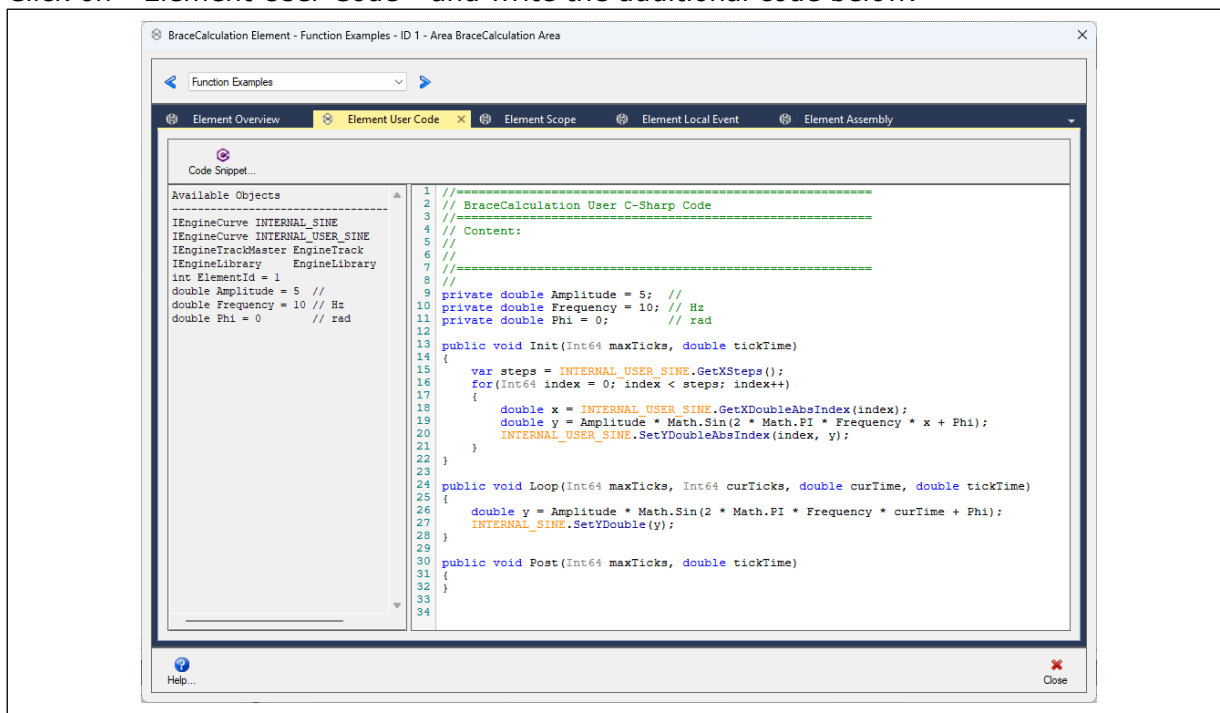


Figure 11 Elements User Code

Now everything is defined and coded.

In the BraceCalculation main window:

- Click <Save> to store the work.
- Click <Build> to compile the user code.
- Click <Start> to start the calculation.

Everything should be fine without any Errors in the build event window.

Click on <Element Scope> in the Element Dialog.

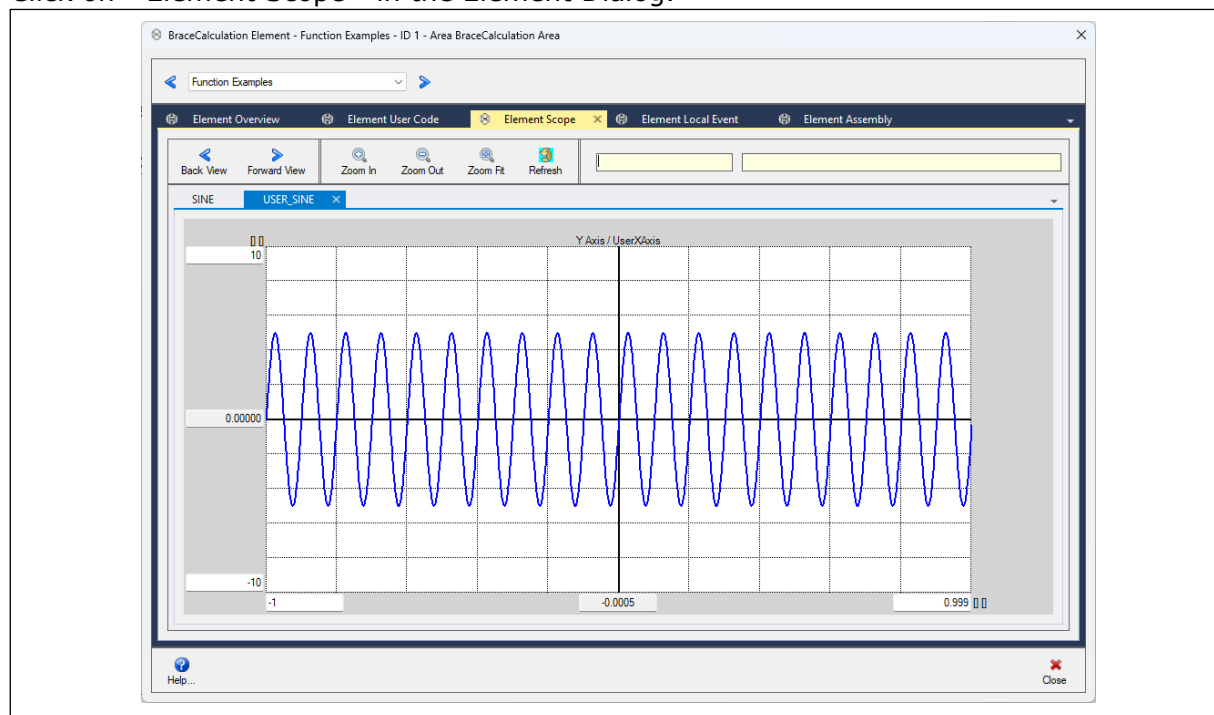


Figure 12 Elements USER_SINE Scope

3 How BraceCalculation Works

3.1 Overview

BraceCalculation is a software package for modeling, calculating and simulating mathematical systems.

Like in other software packages, it is necessary to understand what and how you want to do things in BraceCalculation. That also requires a mathematical and physical understanding of the modeled system.

A system can be thought of as a black box. It contains input signals and output signals. The system can be divided into sub-systems which can relate to other sub-systems. In addition, a system or a sub-system contains internal states and information.

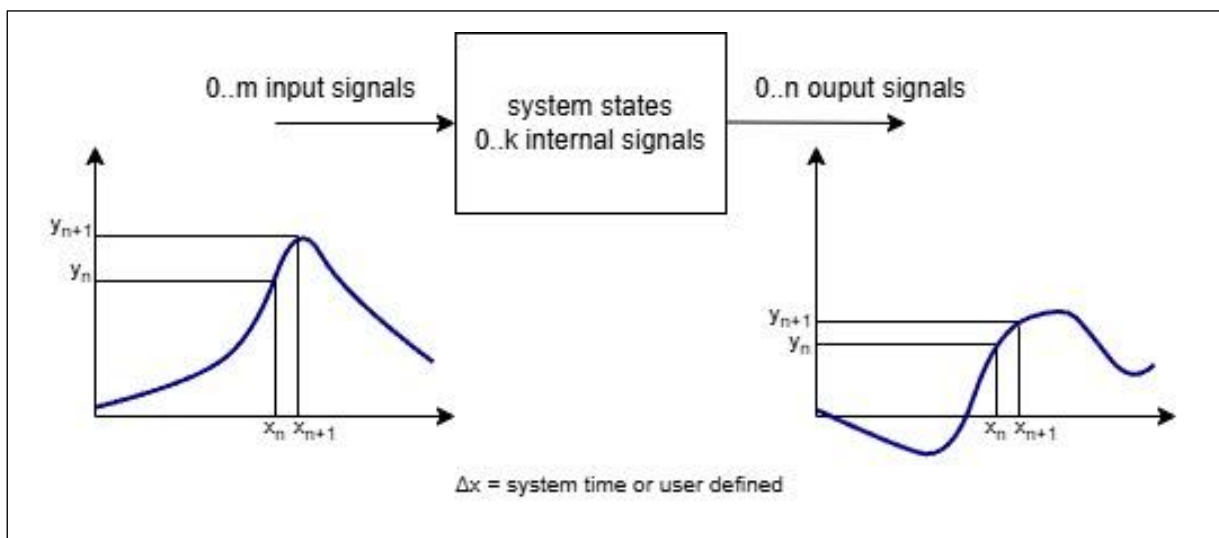


Figure 13 System Input and Output Signals

Signals can be dependent on other signals in terms of value and time e.g. integration, differentiation.

Signals can be complete independently e.g. filter curve, diode characteristic curve.

3.2 Signal Array

Every BraceCalculation signal has a defined length and type with a discrete step width. This step can be defined as system time step width or as user defined step width with any unit.

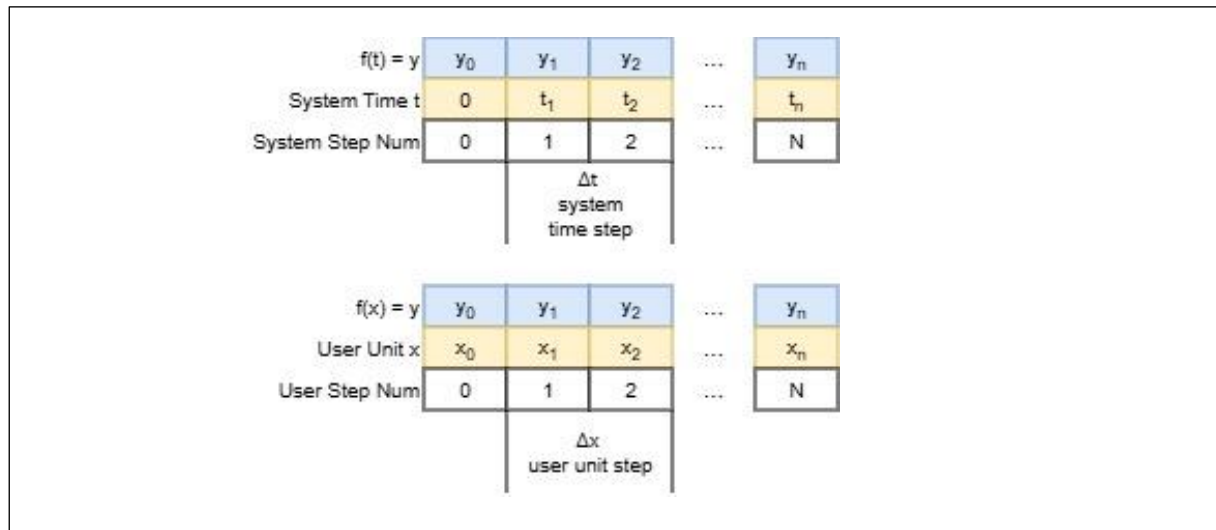


Figure 14 Signal Array

The minimal array size is 10. The maximum array size depends on the available PC memory. In addition, the calculation time needed depends on the number of signal arrays and the array size.

3.3 Calculation

BraceCalculation calculates the signal arrays in three steps:

- Within Initialization process (one time call at calculation start).
- Within Loop process (calls according to number of system steps).
- Within Post process (one time call at calculation end).

The calculation order within these processes is determined by the user.

In general, the following applies:

- Initialization and Post processes are used for user defined signal arrays.
- Loop process is used for system time defined signal arrays.

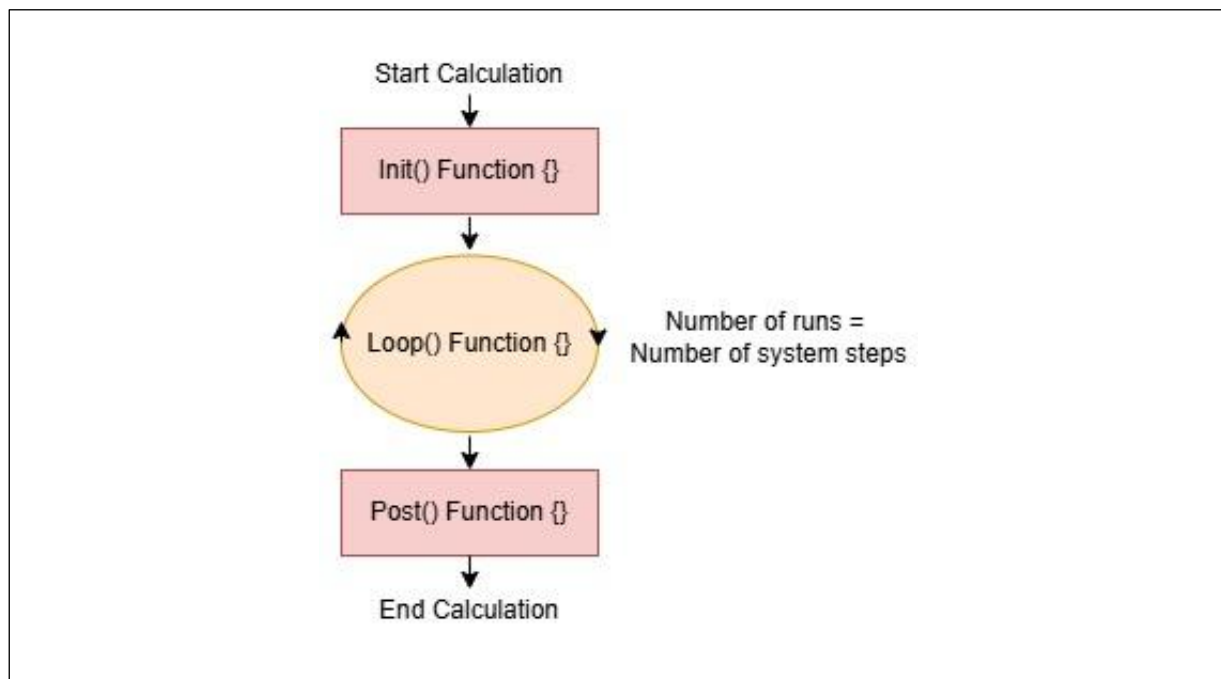


Figure 15 Calculation Thread with Init(), Loop() and Post()

Every BraceCalculation Element contains these functions. The user can program the content themselves using C# programming language.



3.4 Function and Signal Behavior

The signal array type, length and step width is defined according to the curve definitions in the BraceCalculation settings.

The signal array values and behavior are according to the user C# programming.

Example:

- System Time Curves, one second, one microsecond step
 - Y axis type FLOAT64 (double precision)
 - Leads to zero-based signal array with one million steps from $t_{\min}$ 0.0 to $t_{\max}$ 0.999999 second.
- User C# programming:
 - $y1(t) = 50V \cdot \sin(2\pi \cdot 100Hz \cdot t)$
 - $y2(t) = 10V \cdot \cos(2\pi \cdot 100Hz \cdot t)$
 - $y3(t) = y1(t) + y2(t)$

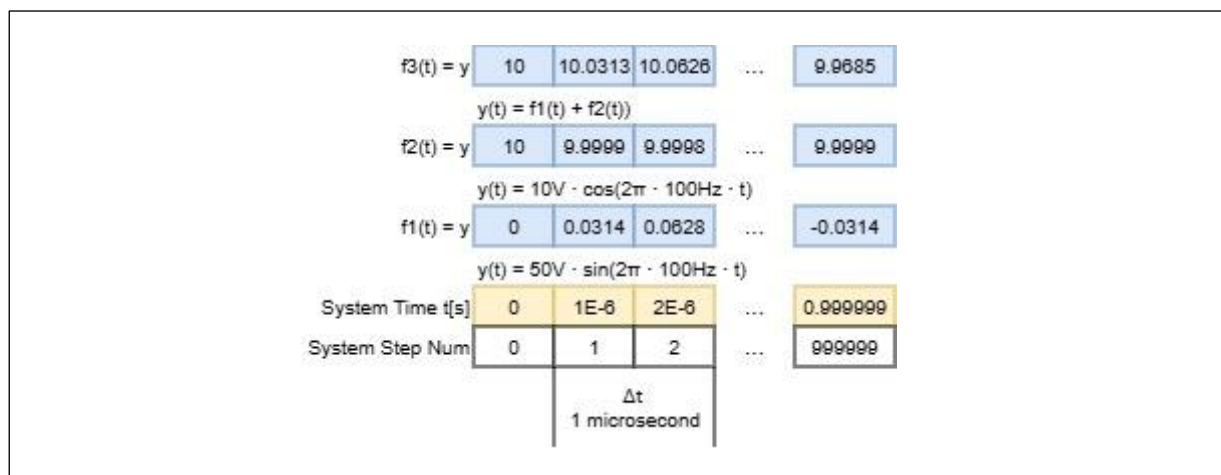


Figure 16 Signal Array Calculation



3.5 Controlling Tracks

With BraceCalculation Tracks, you can transmit and receive controlling data from one Element to another.

You can expand this functionality to user assemblies to transmit and receive these data.

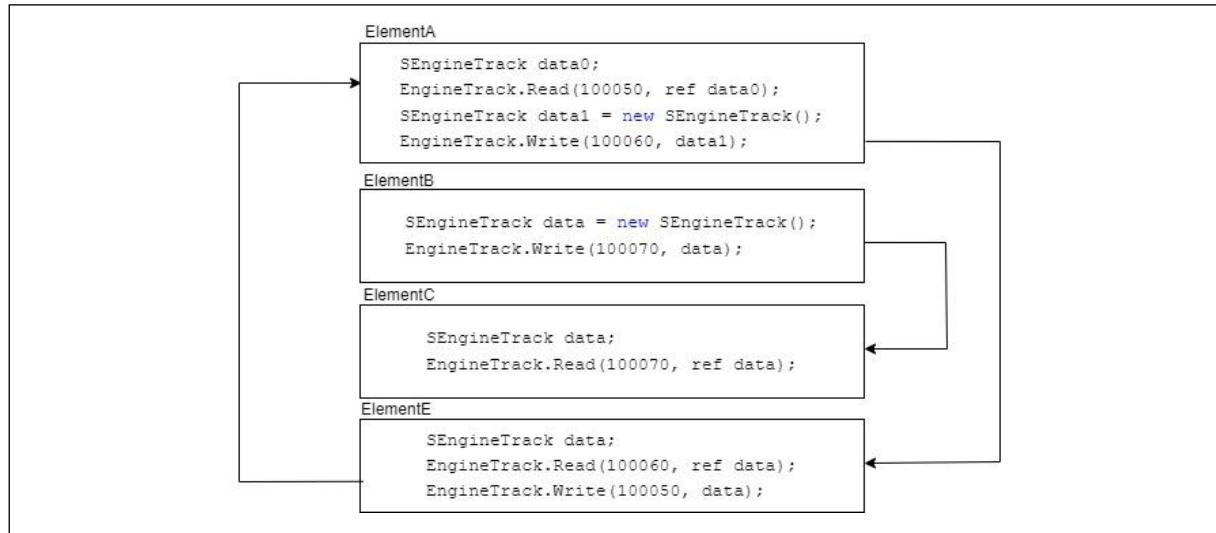


Figure 17 Element TRACK communication example.

4 Using BraceCalculation

4.1 Introduction

Create your system with the graphical schematic area of BraceCalculation.
Write you own C# code, build and start the calculations.

4.2 Create a new Environment

Create a new *.xbcalc and <save as...> it with your own system name.

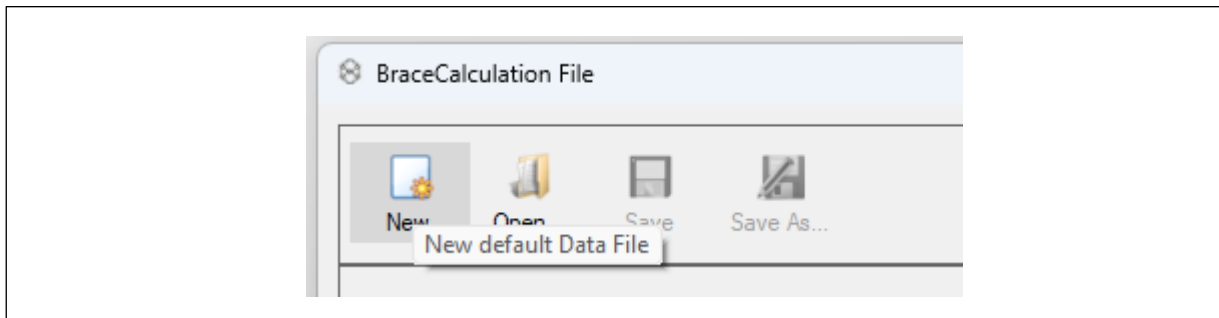


Figure 18 New empty environment file

4.3 Create a new System

Create new parts for your system in BraceCalculation.

- Areas
- Elements
- Pins
- Curves
- Wires
- Tracks

And define the behavior of your system.

- Calculation timing
- Element user C# code.
- Input Pins, Output Pins, Internal Pins
- System time Curves, user Curves
- Calculation timing

4.3.1 Create Drawing Areas

BraceCalculation creates automatically one main drawing area.

If you would like to structure your system more, you can create additional areas.

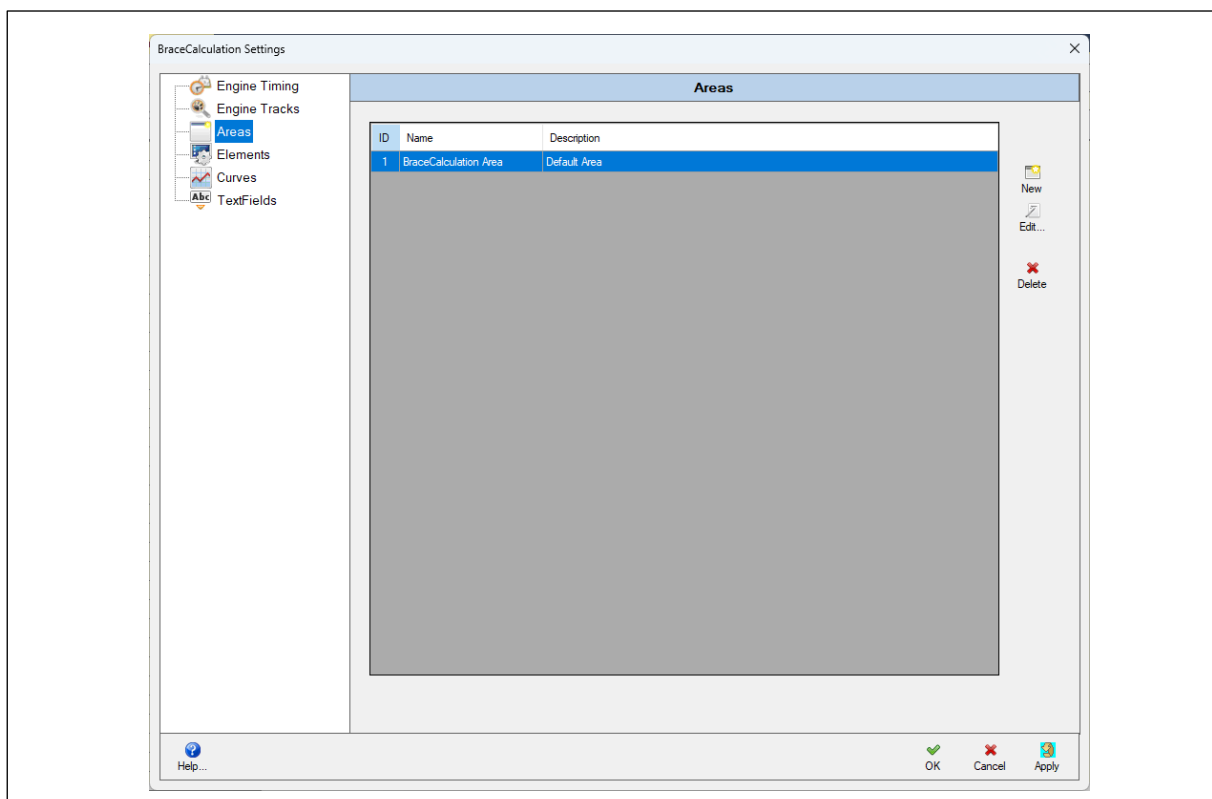


Figure 19 Create new Area in the Settings dialog

Remarks

- You need a professional version to create additional areas.

4.3.2 Create Timing Frame

Before you set up the system, you should know the time frame you want to simulate. Of course, you can change this later.

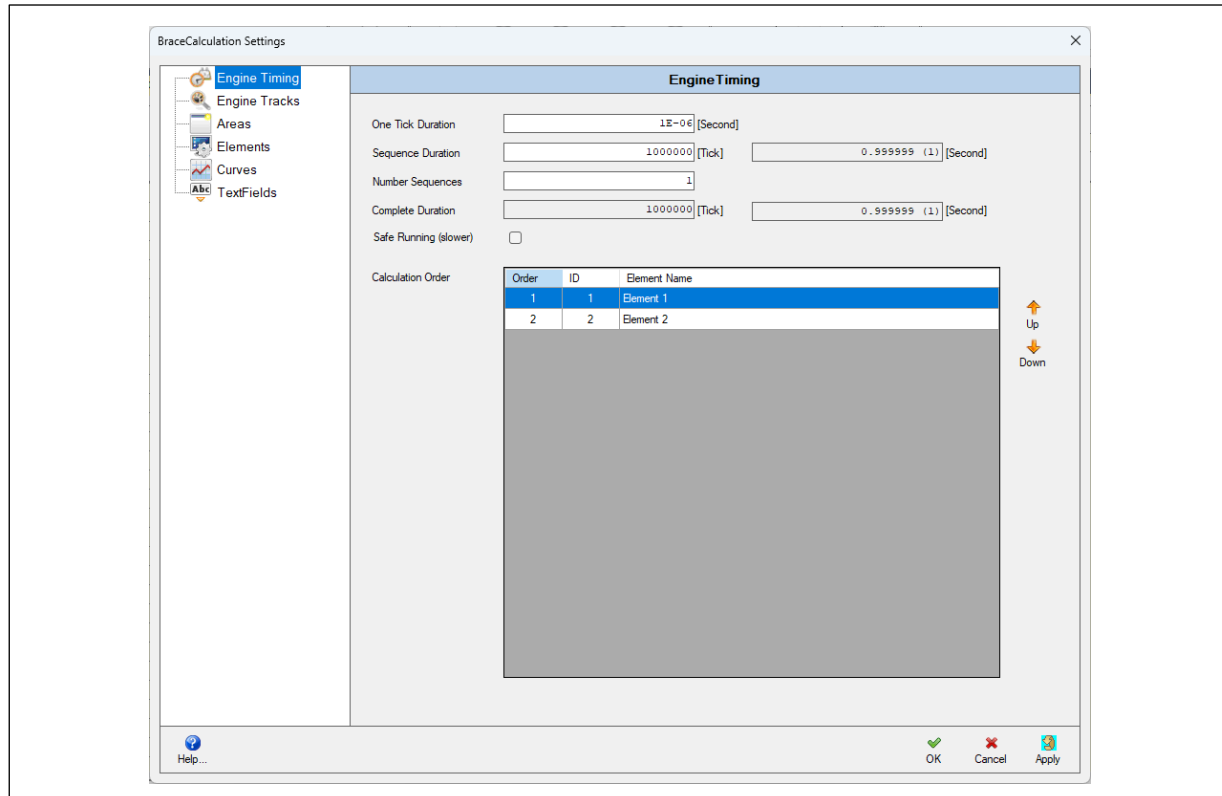


Figure 20 Default timing frame with $1\mu s$ tick duration and 1s calculation time.

BraceCalculation creates all arrays for the length of one sequence. If you define more than one sequence, the arrays will be overwritten consecutively.

4.3.3 Create new Elements

Elements are the main parts of every system. It provides the environment with a wide variety of signal definitions and signal behavior.

At least one Element is necessary for a BraceCalculation system.

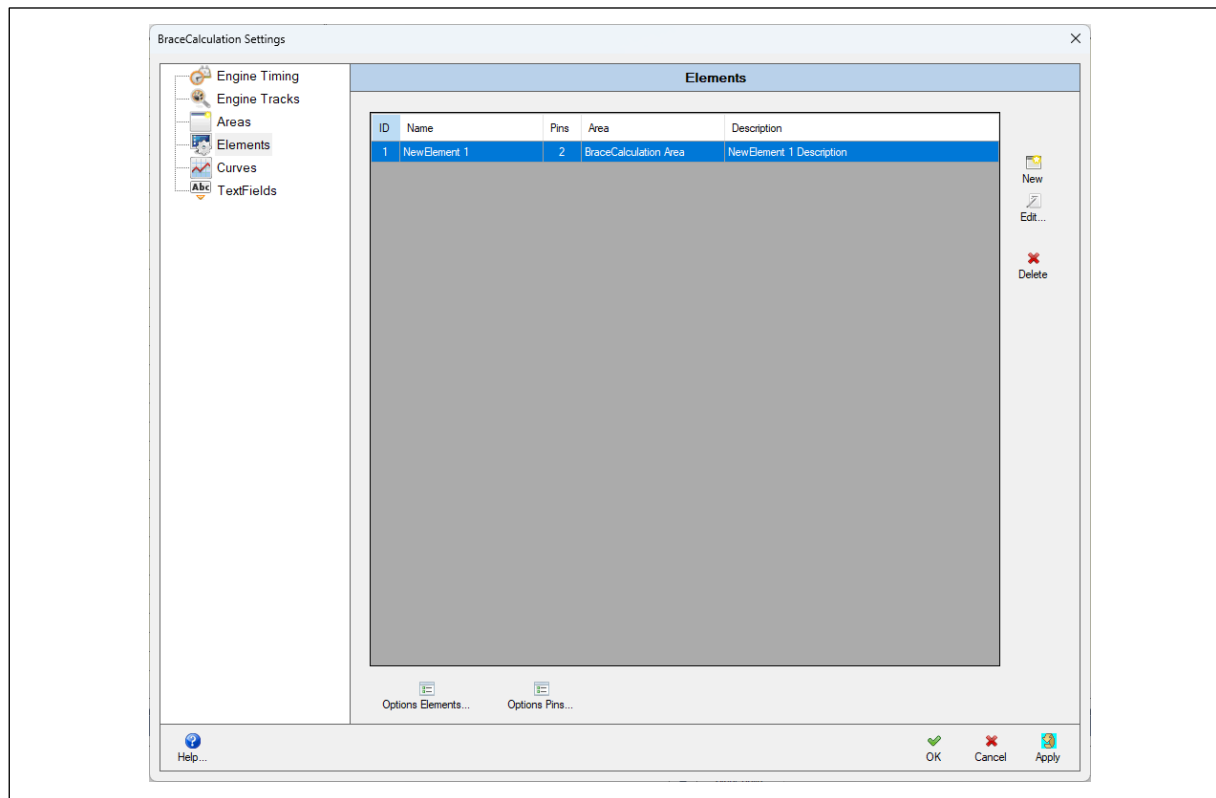


Figure 21 Create new Element in the Settings dialog

Create a proper number of Elements to model your system. The height of the Element defines the maximum number of possible input/ output Pins. The number of internal Pins is not limited to the Element height.

For example: If you have an electrical circuit, you can structure the Elements according to the electronic parts.

- Element for a voltage source
- Element for a resistor
- Element for an inductance
- Element for a capacitor
- Element for operation

4.3.4 Create new Pins

Pins are necessary to bind them to a Curve. Every Element contains several Pins. You cannot use a Curve without a Pin

- Element Output Pin
- Element Input Pin
- Element Internal Pin

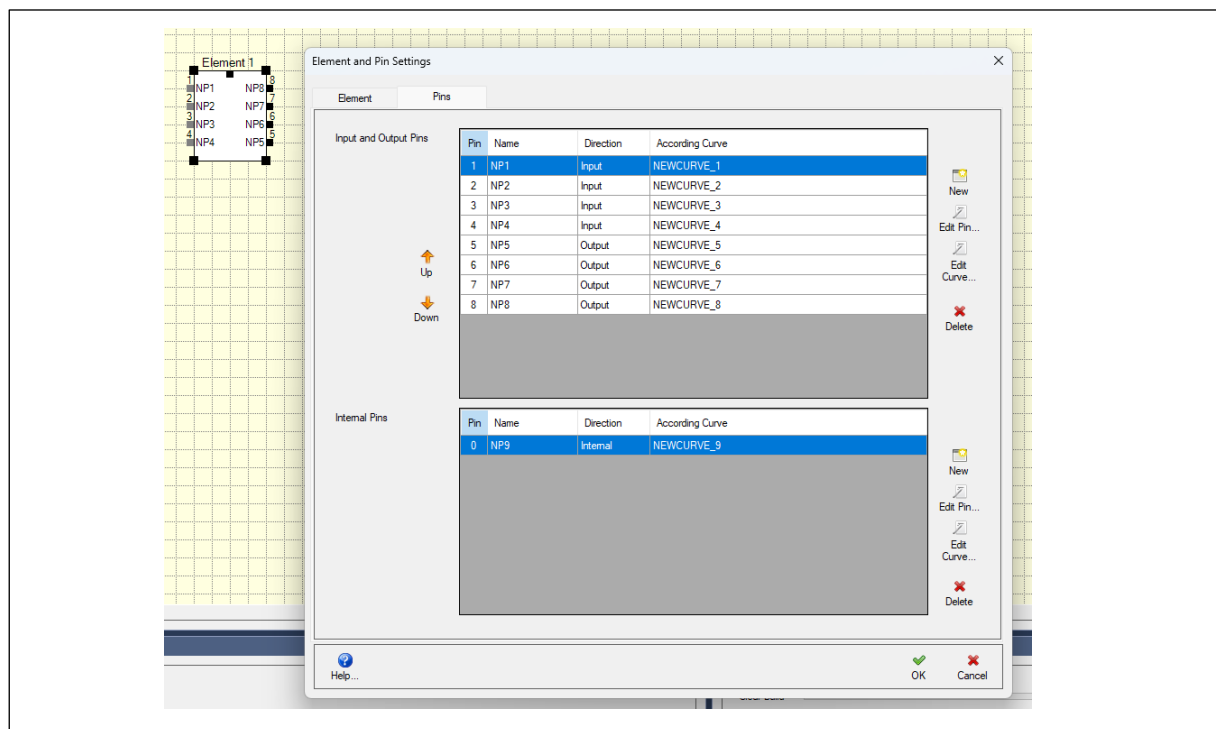


Figure 22 Element with 4 Input Pins and 4 Output Pins and 1 Internal Pin

4.3.5 Create new Curves

Curves are the second most important thing after the Elements in BraceCalculation. Unlike Pins, one Curve is related to one or more Elements. The connection part is the related Element Pin. A Curve works as connecting link between the Elements.

As we have seen, a Curve consists of arrays of values of arbitrary length and different types.

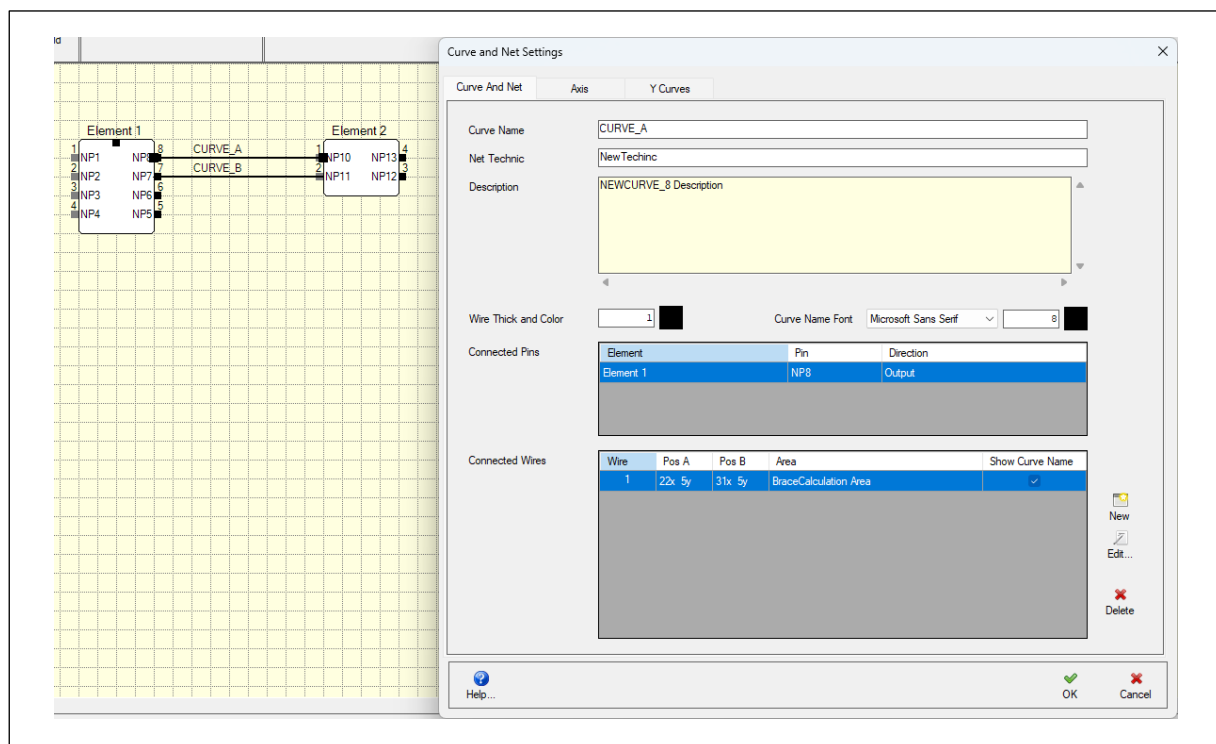


Figure 23 Curve definition relates to Pins and Wires.



Curve X Axis

Parameters/ Base	System Time	System Tick	User
Number Arrays	1	1	1
X size and unit	Time [second]	Ticks [ti]	User defined
Start	Timing Start	0	User defined
End	Timing End	Timing Ticks	User defined
Steps	Timing Steps	1	User defined
Scale	Linear	Linear	Linear Logarithmic
Type	FLOAT64 (double)	SINT64	FLOAT64 SINT64
Loop Calculation	✓	✓	-
Init Calculation	-	-	✓
Post Calculation	-	-	✓

Curve Y Axis

Parameters/ Base	System Time	System Tick	User
Number Arrays	1, 2 ... n ¹	1, 2 ... n ¹	1, 2 ... n ¹
Y size and unit	User defined	Ticks [ti]	User defined
Start	User defined	0	User defined
End	User defined	Timing Ticks	User defined
Steps	User defined	1	User defined
Scale	Linear	Linear	Linear Logarithmic
Type	FLOAT64 (double) SINT64 BYTEARRAY	FLOAT64 SINT64 BYTEARRAY	FLOAT64 SINT64 BYTEARRAY
Loop Calculation	✓	✓	-
Init Calculation	-	-	✓
Post Calculation	-	-	✓

¹ More than 2 Y Curve arrays require the Professional version of [BraceCalculation](#).

4.3.6 Draw Wires

Every Curve can be related to some graphical wires.

But and that is important, **it doesn't have to.**

A wireframe is a graphical tool that helps to better understand the relationships between elements.

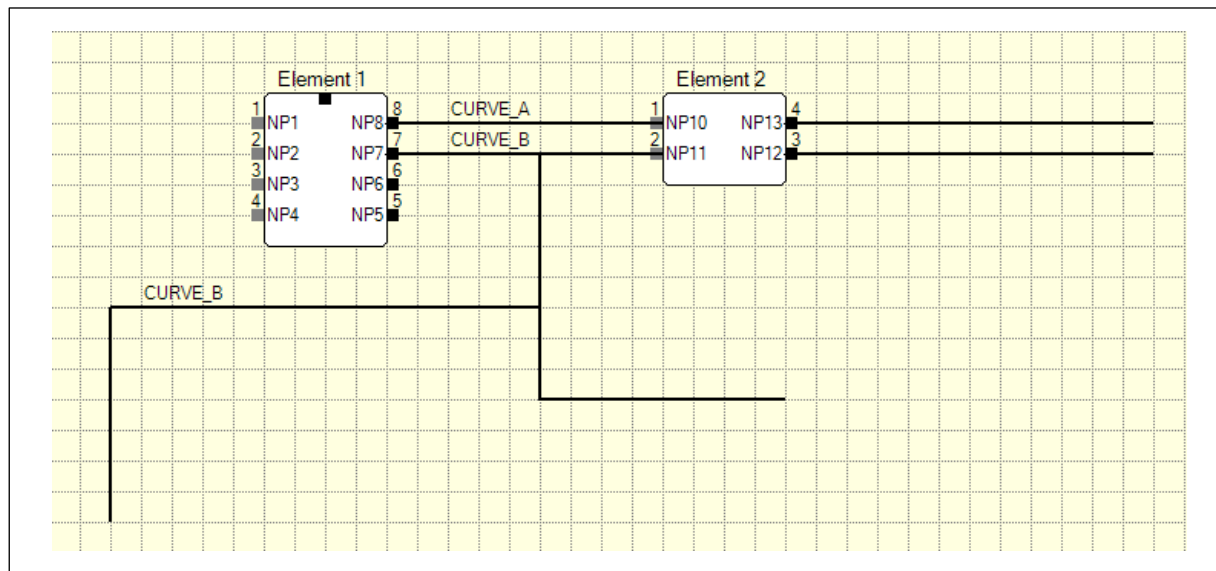


Figure 24 Draw Wires by the mouse.

You can change all wire settings by the Wire dialog.

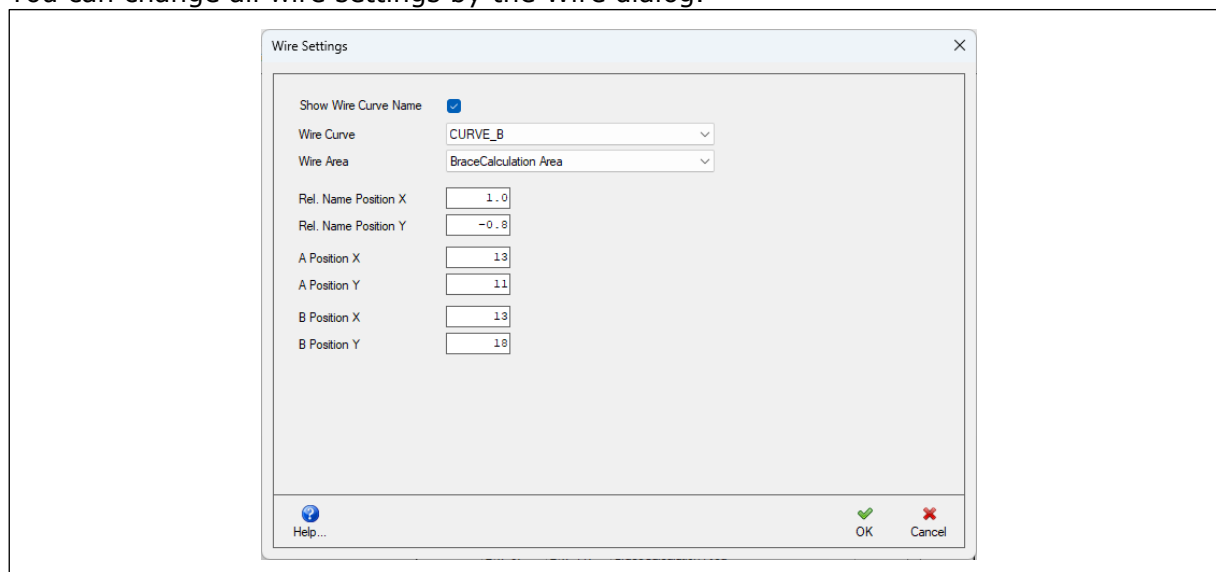


Figure 25 Wire dialog for one piece of wire.

4.4 Write your Element Code

Until now, we have defined our system frame.

You can already build and compile the system without any user code.

You can already start and stop your calculation and simulation.

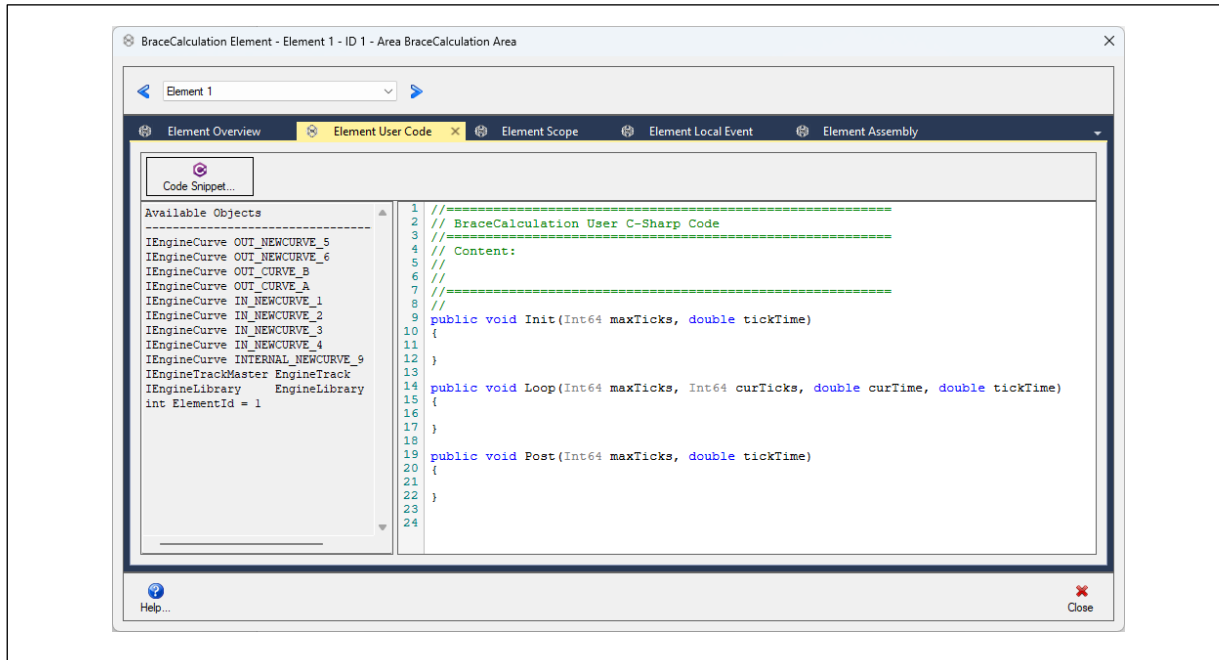


Figure 26 Code window with code frame for user programming.

If you build your system first, you will see the available objects on the left.

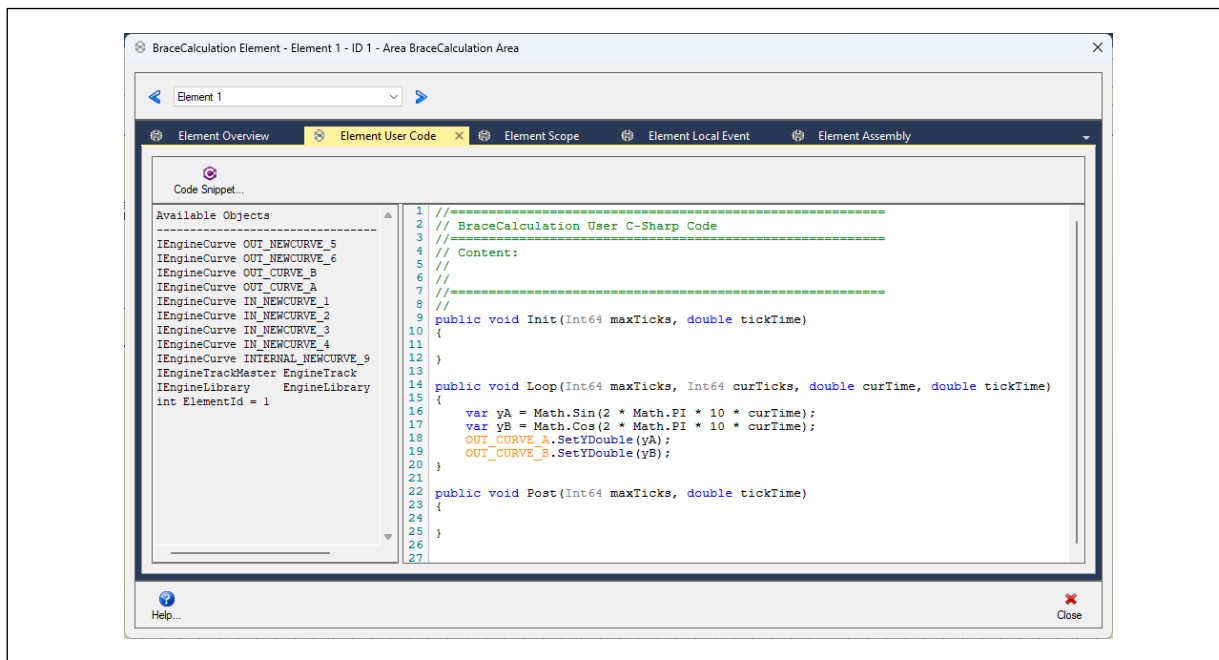


Figure 27 Code window with user code for CURVE_A and CURVE_B.

4.5 Build and Compile System

Now your system is prepared.

Press the <Build> button to compile your added C# code together with the source environment.

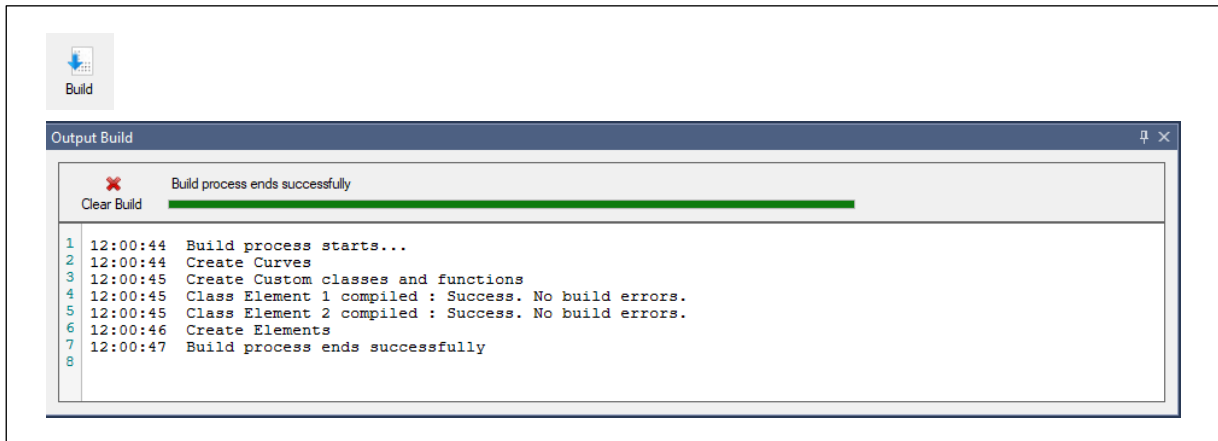


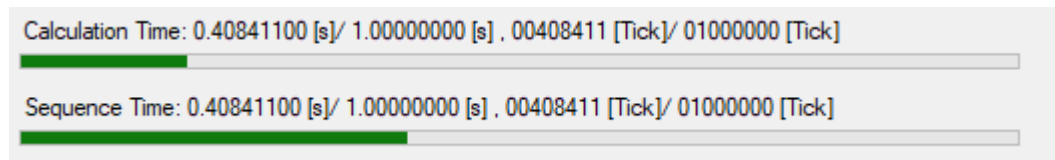
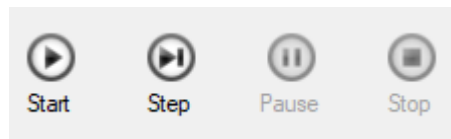
Figure 28 Build process window.

If your code is faulty, you will get a red error message.



4.6 Calculate and Simulate System

Now you can run the calculation of your two-curve system.



Sequence after pushing Start button:

- Begin of the calculation with Element User code Init() calls.
- The order of the calls is according to the Element calculation order in the settings.
- After all Init() the Loop() calls are performed at system time 0 and go through until the system time end is reached.
- After that all Post() calls are performed according to the Element calculation order.

Sequence pushing Step button:

- Start or continue the calculation.
- One step performs one function call (Init, Loop or Post).

Sequence pushing Pause button:

- The calculation holds on after the actual function call (Init, Loop, Post).
- You can Stop, Step or Start from the current position.

Sequence pushing Stop button:

- The calculation stops and calls all Post() functions to finish it.

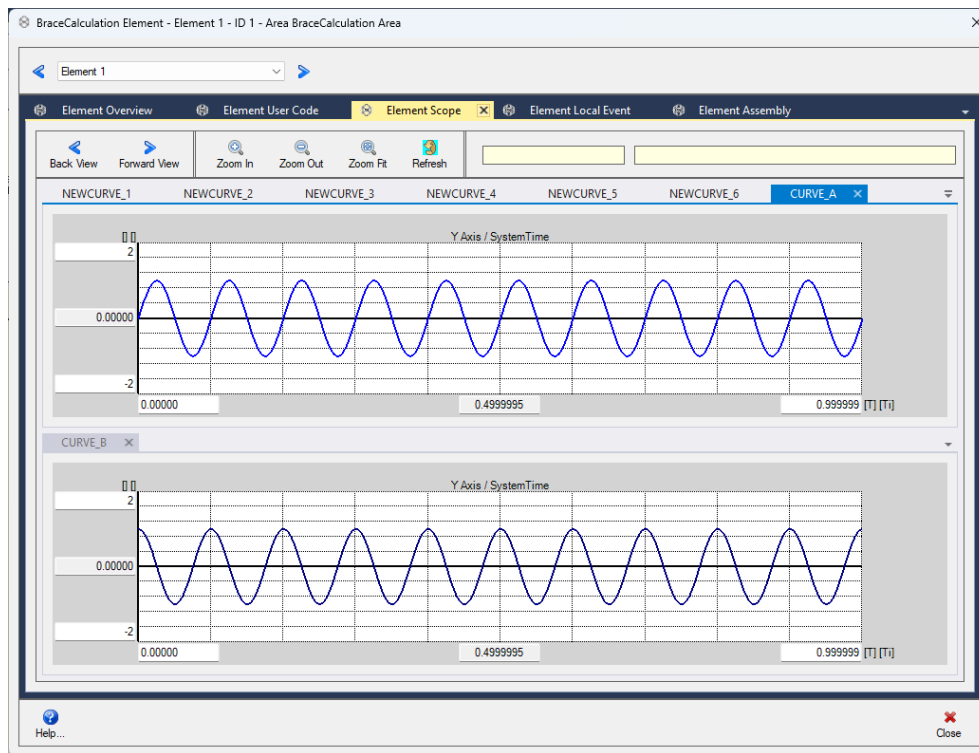


Figure 29 Code window with user code for CURVE_A and CURVE_B.

After or during long calculation, you can look at the Element scope to see every curve as graphical plot.

4.7 System Control with Tracks

Curves and Pins connected the Elements together.

There is a second way to send data between the Elements. This can be done by Tracks.

Tracks don't use any Curve data. Tracks transport data from a static C# structure.

In every Element, Tracks data can be received or transmitted from/ to another Element within the user C# code.

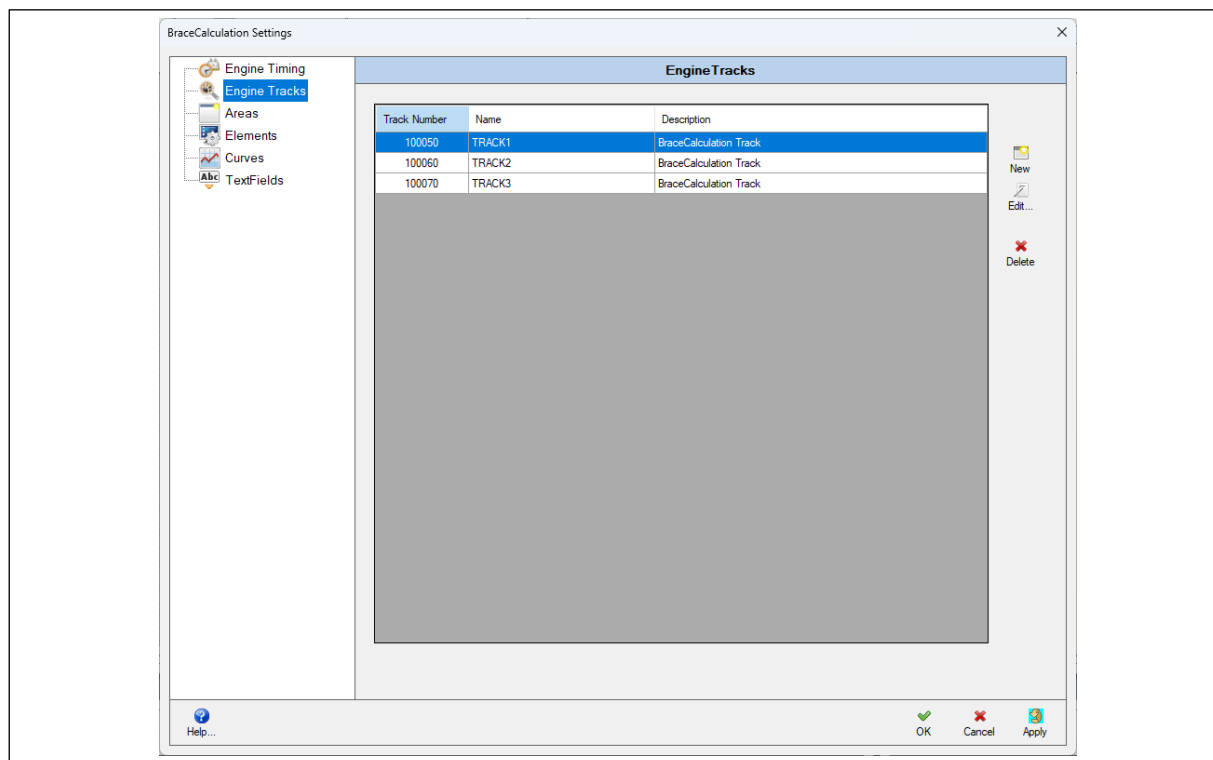


Figure 30 TRACK settings.

4.8 Write an Assembly

Additional to the Tracks, you can write your own C# assembly and communicate with the Elements through Tracks.

Your correctly created Assembly, you can include one per Element.

For details see examples.

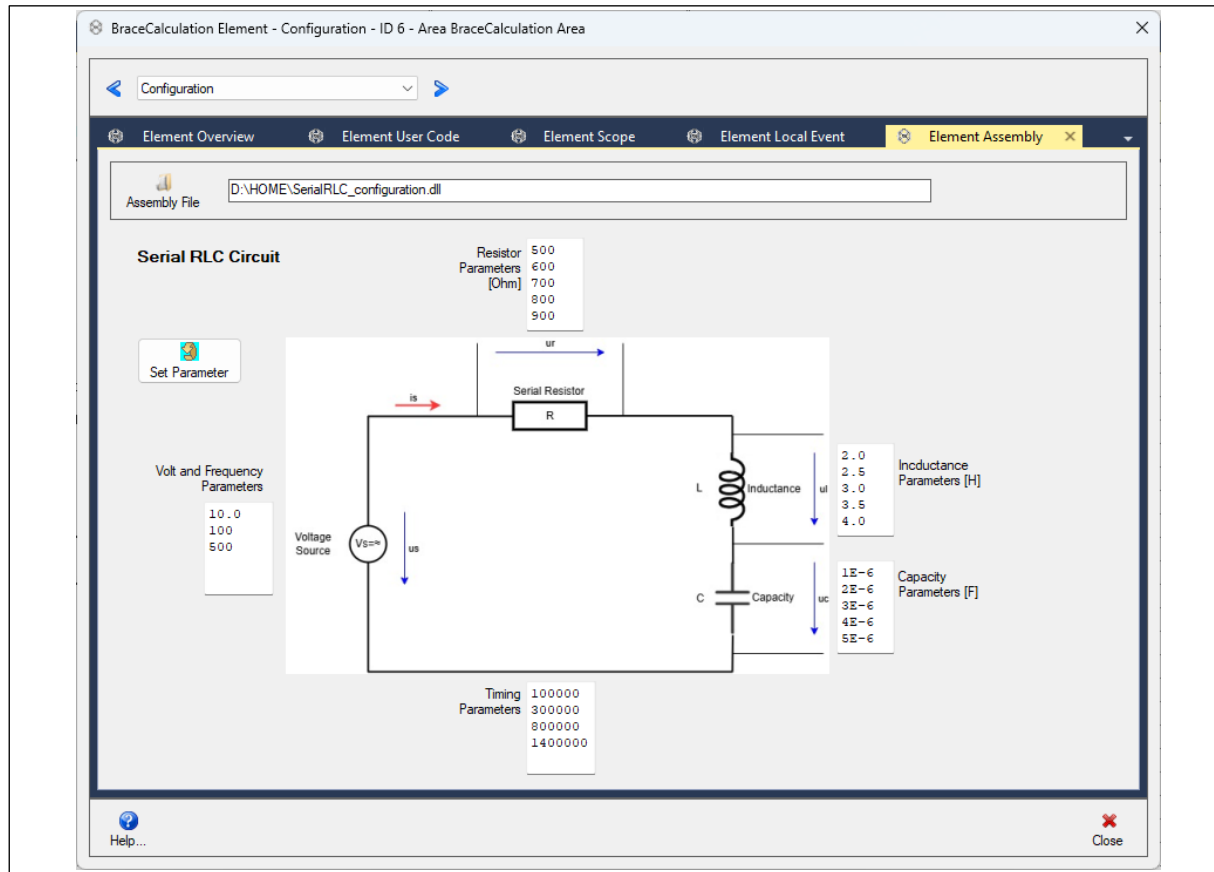
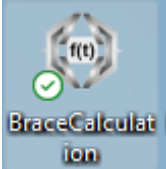
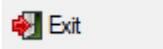


Figure 31 User Assembly include. Example of a RLC circuit to set different parameters.

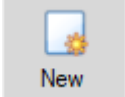
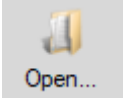
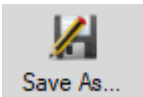
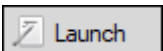
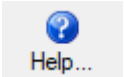


5 Reference

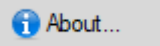
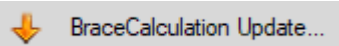
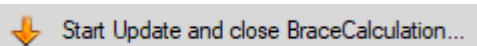
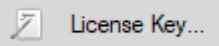
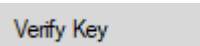
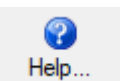
5.1 Start and Exit

Item	Action, Command	Precondition	Description
Start BraceCalculation 	Run brace_calc.exe	Correct installed BraceCalculation Application	BraceCalculation starts and shows the File Dialog.
Start BraceCalculation	Double-click *.xbcalc file	Correct installed BraceCalculation Application. File extension XBCALC must be assigned to the application.	BraceCalculation starts with the xbcalc file and loads the file data.
Exit BraceCalculation  	Click <Exit> or Click Top-Right	BraceCalculation is running with saved user data.	BraceCalculation stops running.

5.2 File Handling

Item	Action, Command	Precondition	Description
New XBCALC File 	Click <New> in File dialog.	BraceCalculation is running	New and empty XBCALC file is generated and loaded with a default name.
Open... XBCALC File 	Click <Open...> in File dialog.	BraceCalculation is running	Open and load stored XBCALC file.
Save XBCALC File 	Click <Save> in Main window or File dialog.	BraceCalculation is running	Save data to open XBCALC file.
Save As... XBCALC File 	Click <Save As...> in File dialog.	BraceCalculation is running	Save data to chosen XBCALC file.
Load XBCALC data 	Click <Launch>	BraceCalculation is running. Valid recent file.	Data is loading from selected recent file.
Close File Dialog 	Click <Close>	BraceCalculation is running	Close File Dialog without any (new) data loading.
Open Documentation 	Click <Help...>	BraceCalculation is running	Opens the Help Dialog with the BraceCaclulation documentation.

5.3 About BraceCalculation

Item	Action, Command	Precondition	Description
Open About Dialog 	Click <About...>	BraceCalculation is running	Open BraceCalculation About Dialog for impressum, license and updates.
Open Update Dialog 	Click <BraceCalculation Update...>	Update available	Dialog contains information about an update.
Run Update Process 	Click <Start Update and close BraceCalculation ...>	Update available	Update process starts and closes running BraceCalculation.
License Key 	Click <License Key...>	No Data loaded.	Input, verify BraceCalculation license key for Professional version.
Verify Key 	Click <Verify Key>	No Data loaded.	6x5 character key is verifying.
Close File Dialog 	Click <Close>	BraceCalculation is running	Close About Dialog
Open Documentation 	Click <Help...>	BraceCalculation is running	Opens the Help Dialog with the BraceCaclulation documentation.

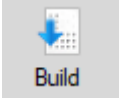
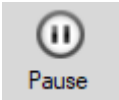


5.4 Help

Item	Action, Command	Precondition	Description
Open Documentation  Help...	Click <Help...>	BraceCalculation is running	Opens the Help Dialog with the BraceCaclulation documentation.

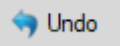
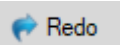
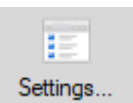
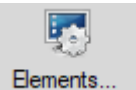


5.5 Calculation Controls

Item	Action, Command	Precondition	Description
Build 	Click <Build>	Data loaded.	Compiles and build Elements C# code.
Start Calculation 	Click <Start>	Data loaded and built.	Start calculation according to Elements and Timing.
Step Calculation 	Click <Step>	Data loaded and built. Active calculation isn't running.	Calculate one step according to Elements and Timing.
Pause Calculation 	Click <Pause>	Data loaded and built. Active calculation is running.	Halt the running calculation.
Stop Calculation 	Click <Stop>	Data loaded and built. Active calculation is running.	Stop the running calculation.
Calculation Progress 		Data loaded and built.	Show calculation progress.

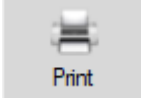
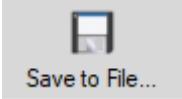
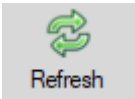


5.6 Additional Controls

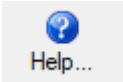
Item	Action, Command	Precondition	Description
Undo 	Click <Undo>	Done area drawing action.	Undo performed drawing action.
Redo 	Click <Redo>	Done area drawing action.	Redo performed drawing action.
Settings Dialog 	Click <Settings...>	Data loaded.	Open Settings dialog.
Elements Dialog 	Click <Elements...>	Data loaded.	Open Elements dialog.



5.7 Outputs

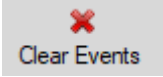
Item	Action, Command	Precondition	Description
Print Sheets 	Click <Print>	Data loaded.	Print all selected sheets and pages from the dialog.
Save to File... 	Click <Save to File...>	Data loaded.	Save all selected sheets and pages to a PDF, JPEG, PNG or BMP format file.
Refresh View 	Click <Refresh>	Data loaded.	Refresh the view area after changing properties.
Turn back one output page 	Click <Back>	Data loaded.	Show back sheet in the view.
Turn forward one output page 	Click <Forward>	Data loaded.	Show forward sheet in the view.
Title Field	Input title text	Data loaded.	Title text for selected sheets.
Date Field	Select Date	Data loaded.	Date text for selected sheets.
Creator Field	Input name	Data loaded.	Creator text for selected sheets.
Preview Size Field	Select Format	Data loaded.	Preview format for selected sheets.
Sheet List 	Click Checkbox	Data loaded.	Select, unselect available sheets.
Close Outputs Dialog 	Click <Close>	Data loaded.	Close Outputs Dialog



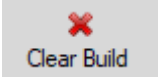
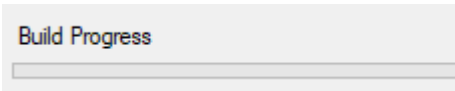
Open Documentation 	Click <Help...>	Data loaded.	Opens the Help Dialog with the BraceCaclulation documentation.
---	-----------------	--------------	--



5.8 Output Events

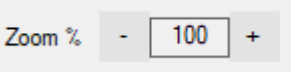
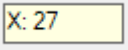
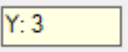
Item	Action, Command	Precondition	Description
Clear Events 	Click <Clear Events>	Data loaded.	Clear output window event text.

5.9 Output Build

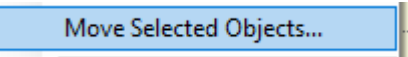
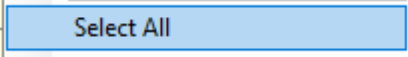
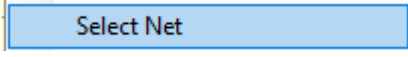
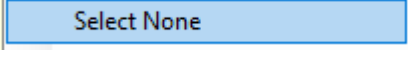
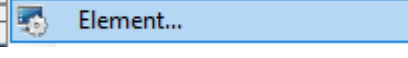
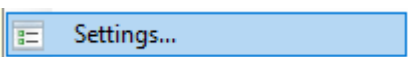
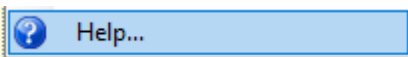
Item	Action, Command	Precondition	Description
Clear Build 	Click <Clear Build>	Data loaded.	Clear output window build text.
Build Progress 		Data loaded.	Shows the progress of the build process.



5.10 Schematic Area

Item	Action, Command	Precondition	Description
Cut 	Click <Cut> Area Click <Cut> Context Menu CTRL + X	Area part selected.	Cut the selected area object to the clipboard.
Copy 	Click <Copy> Area Click <Copy> Context Menu CTRL + C	Area part selected.	Copy the selected area object to the clipboard.
Delete 	Click <Delete> Area Click <Delete> Context Menu DELETE	Area part selected.	Delete the selected area object.
Paste 	Click <Paste> Area Click <Paste> Context Menu CTRL + V	Area part is in Clipboard.	Paste the stored area object from the clipboard.
Add Element 	Click <Add Element> Area Click <Add Element> Context Menu	Data loaded.	Add new empty Element to the current area.
Add TextField 	Click <Add TextField> Area Click <Add TextField> Context Menu	Data loaded.	Add new empty TextField to the current area.
Zoom 	Click <-> or <+>	Data loaded.	Set the zoom from 25% to 500%.
Zoom Fit 100% 	Click <Zoom Fit 100%>	Data loaded.	Set zoom to 100%.
X Cursor 	Move mouse cursor over the area	Data loaded.	Show X coordinate grid units.
Y Cursor 	Move mouse cursor over the area	Data loaded.	Show Y coordinate grid units.



Part Cursor 	Move mouse cursor over the area	Data loaded.	Show area part if cursor over it.
Move Selected Objects 	Click <Move Selected Objects> Context menu.	Data loaded	Move selected object to another area tab.
Select All 	Click <Select All> Context menu.	Data loaded.	Select all objects.
Select Net 	Click <Select All> Context menu.	Data loaded. Wire selected.	Select related Net from a selected wire.
Select None 	Click <Select None> Context menu.	Data loaded.	Unselect all objects.
Element 	Click <Element...> Context menu.	Data loaded.	Show Element dialog of Element under cursor or shows first Element.
Settings 	Click <Settings...> Context menu.	Data loaded.	Show Settings dialog.
Open Documentation 	Click <Help...> Context menu.	Data loaded.	Opens the Help Dialog with the BraceCaclulation documentation.



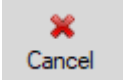
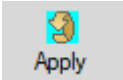
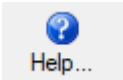
5.11 Schematic Drawing

Item	Action, Command	Precondition	Description
Select Object	Click Object	Data loaded.	Select Object at mouse cursor. Unselect previous selected objects.
Select Objects	Click Object + STRG	Data loaded.	Select Object at mouse cursor.
Select All Objects	CTRL + A	Data loaded.	Select all Objects.
Move Objects	Click Object and hold left mouse button. RIGHT, LEFT, UP, DOWN	Object selected.	Move the selected Objects one grid unit.
Connect new Wire at Pin	Click Pin and Drag	Element Pin	Add and connect a new wire to the selected Pin.
Connect new Wire at Wire	Click Wire + CTRL and Drag	Wire	Add and connect a new wire to the selected in perpendicular.



5.12 Settings

5.12.1 General

Item	Action, Command	Precondition	Description
OK 	Click <OK>	Data loaded.	Accept the modified data to the current loaded data and close the dialog.
Cancel 	Click <Cancel>	Data loaded.	Close the dialog without accepting the modified data.
Apply 	Click <Apply>	Data loaded.	Accept the modified data to the current loaded data.
Open Documentation 	Click <Help...>	Data loaded.	Opens the Help Dialog with the BraceCaclulation documentation.

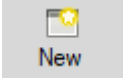
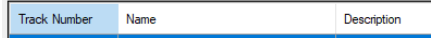


5.12.2 Engine Timing

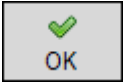
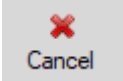
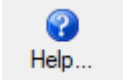
Item	Action, Command	Precondition	Description
One Tick Duration	Input number	Data loaded.	Set the basic tick duration in second as double precision number.
Sequence Duration	Input number	Data loaded.	Set the number of basic ticks for one sequence. (Memory relevant)
Number Sequences	Input number	Data loaded.	Set the number of sequences for the calculation. Only the actual calculation sequence contains usable data. (Not Memory relevant)
Safe Running (slower) 	Checkbox	Data loaded.	Calculation will run in safe mode. Calculation speed will slow down.
Calculation List 		Data loaded.	Show the calculation order of the Elements. Lowest number has highest calculation priority
Up 	Click <Up>	List item selected.	Move selected list item one up for changed calculation priority.
Down 	Click <Down>	List item selected.	Move selected list item one down for changed calculation priority.



5.12.3 Engine Tracks

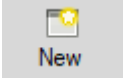
Item	Action, Command	Precondition	Description
New Track 	Click <New>	Data loaded.	Create a new Track.
Edit Track 	Click <Edit>	Data loaded.	Open Track dialog.
Delete Track 	Click <Delete>	Data loaded.	Delete selected Track from List.
Track List 		Data loaded.	See all defined Tracks.

**5.12.3.1 Edit Track**

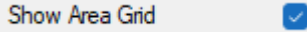
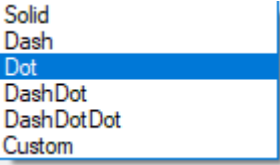
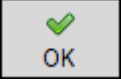
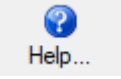
Item	Action, Command	Precondition	Description
Track Number	Input Number	Data loaded. Track defined.	Edit the Track number. Must be > 100000 and must be equal.
Track Name	Input Name	Data loaded. Track defined.	Edit the Track name. Doesn't have to be equal.
Track Description	Input Text	Data loaded. Track defined.	Edit the Track description.
OK 	Click <OK>	Data loaded. Track defined.	Take over the set Track parameters and close the dialog.
Cancel 	Click <Cancel>	Data loaded. Track defined.	Don't take over the set Track parameters and close the dialog.
Open Documentation 	Click <Help...>	Data loaded. Track defined.	Opens the Help Dialog with the BraceCaclulation documentation.



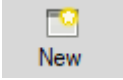
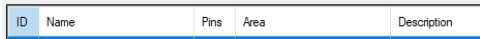
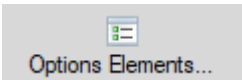
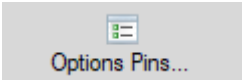
5.12.4 Areas

Item	Action, Command	Precondition	Description
New Area 	Click <New>	Data loaded.	Create a new Area.
Edit Area 	Click <Edit>	Data loaded.	Open Area dialog.
Delete Area 	Click <Delete>	Data loaded.	Delete selected Area from List. Area 1 cannot be deleted.
Area List 		Data loaded.	See all defined Areas.

**5.12.4.1 Edit Area**

Item	Action, Command	Precondition	Description
Area Name	Input Name	Data loaded. Track defined.	Edit the Area name.
Area Description	Input Text	Data loaded. Track defined.	Edit the Area description.
Area Background Color	Click <Color>	Data loaded. Track defined.	Choose background color for this Area.
Show Area Grid 	Click Checkbox	Data loaded. Track defined.	Show grid for this Area.
Area Grid Pattern 	Click Popup	Data loaded. Track defined.	Choose Area grid pattern
Area Grid Units	Input grid units	Data loaded. Track defined.	Choose the number of grid units.
Area Grid Color	Click <Color>	Data loaded. Track defined.	Choose grid color for this Area.
OK 	Click <OK>	Data loaded. Area defined.	Take over the set Area parameters and close the dialog.
Cancel 	Click <Cancel>	Data loaded. Area defined.	Don't take over the set Area parameters and close the dialog.
Open Documentation 	Click <Help...>	Data loaded. Area defined.	Opens the Help Dialog with the BraceCaclulation documentation.

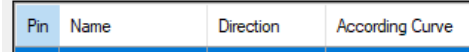
5.12.5 Elements

Item	Action, Command	Precondition	Description
New Element 	Click <New>	Data loaded.	Create a new Element.
Edit Element 	Click <Edit>	Data loaded.	Open Element dialog.
Delete Element 	Click <Delete>	Data loaded.	Delete selected Element from List.
Element List 		Data loaded.	See all defined Elements.
Options Elements 	Click <Options Elements...>	Data loaded.	Open options elements dialog.
Options Pins 	Click <Options Pins...>	Data loaded.	Open options pins dialog.

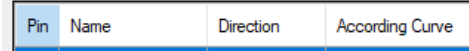
**5.12.5.1 Edit Element**

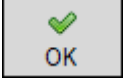
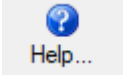
Item	Action, Command	Precondition	Description
Element Name	Input Name	Data loaded. Element defined.	Edit the Element name. No special characters are allowed.
Element Area	Popup Belongs to Area	Data loaded. Element defined.	Element belongs to this Area.
Element Description	Input Text	Data loaded. Element defined.	Edit the Element description.
User Assembly 	Input Filename	Data loaded. Element defined.	Choose own .NET assembly for this Element.
Max. Number of Pins 2..256	Popup Number	Data loaded. Element defined.	Choose the Element height according to number of Pins.
Element Width 2..100	Input grid units	Data loaded. Element defined.	Element width.
Element X Position 0..1000	Input grid units	Data loaded. Element defined.	Element Area X position.
Element Y Position 0..1000	Input grid units	Data loaded. Element defined.	Element Area Y position.
Element Background Color	Click <Color>	Data loaded. Element defined.	Choose Element Background color.



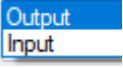
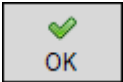
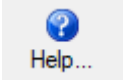
Item	Action, Command	Precondition	Description
New Input/ Output Pin 	Click <New>	Data loaded. Element defined.	Create a new Pin.
Edit Input/ Output Pin 	Click <Edit Pin>	Data loaded. Element defined.	Open Pin dialog.
Edit Curve according to Pin 	Click <Edit Curve>	Data loaded. Element defined.	Open Curve dialog.
Delete Input/ Output Pin 	Click <Delete>	Data loaded. Element defined.	Delete selected Pin from List.
Input/ Output Pin List 		Data loaded. Element defined.	See all defined Input/ Output Pins.
Up 	Click <Up>	List item selected.	Move selected list item one up for changed Pin number.
Down 	Click <Down>	List item selected.	Move selected list item one down for changed Pin number.



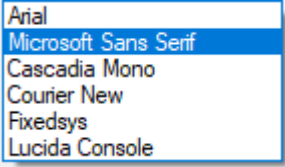
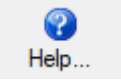
Item	Action, Command	Precondition	Description
New Internal Pin 	Click <New>	Data loaded. Element defined.	Create a new Pin.
Edit Internal Pin 	Click <Edit Pin>	Data loaded. Element defined.	Open Pin dialog.
Edit Curve according to Pin 	Click <Edit Curve>	Data loaded. Element defined.	Open Curve dialog.
Delete Internal Pin 	Click <Delete>	Data loaded. Element defined.	Delete selected Pin from List.
Internal Pin List 		Data loaded. Element defined.	See all defined Internal Pins.

OK 	Click <OK>	Data loaded. Element defined. Pin defined.	Take over the set Element parameters and close the dialog.
Cancel 	Click <Cancel>	Data loaded. Element defined. Pin defined.	Don't take over the set Element parameters and close the dialog.
Open Documentation 	Click <Help...>	Data loaded. Element defined. Pin defined.	Opens the Help Dialog with the BraceCaclulation documentation.

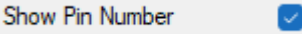
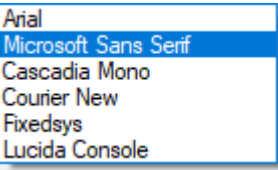
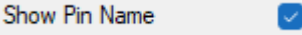
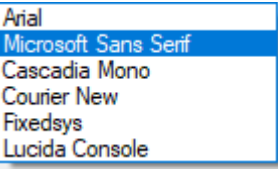
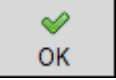
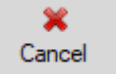
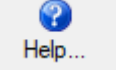
**5.12.5.2 Edit Pin**

Item	Action, Command	Precondition	Description
Pin Name	Input Name	Data loaded. Element defined. Pin defined.	Edit the Pin name. No special characters are allowed.
Pin Direction 	Popup Pin Direction	Data loaded. Element defined. Pin defined.	Choose Pin Direction.
Pin Curve	Popup Curves	Data loaded. Element defined. Pin defined.	Choose a defined Curve for this Pin.
Pin Description	Input Text	Data loaded. Element defined. Pin defined.	Edit the Pin description.
OK 	Click <OK>	Data loaded. Element defined. Pin defined.	Take over the set Pin parameters and close the dialog.
Cancel 	Click <Cancel>	Data loaded. Element defined. Pin defined.	Don't take over the set Pin parameters and close the dialog.
Open Documentation 	Click <Help...>	Data loaded. Element defined. Pin defined.	Opens the Help Dialog with the BraceCalculation documentation.

**5.12.5.3 Options Elements**

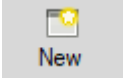
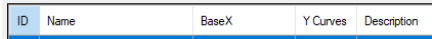
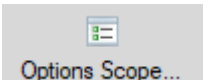
Item	Action, Command	Precondition	Description
Element Frame Color	Click <Color>	Data loaded. Element defined.	Choose Element Frame color.
Element Font 	Popup 	Data loaded. Element defined.	Choose Element Font.
Element Font Height	Input Number	Data loaded. Element defined.	Choose Element Font Height in pixel.
Element Font Color	Click <Color>	Data loaded. Element defined.	Choose Element Font color.
OK 	Click <OK>	Data loaded. Element defined.	Take over the set Option parameters and close the dialog.
Cancel 	Click <Cancel>	Data loaded. Element defined.	Don't take over the set Option parameters and close the dialog.
Open Documentation 	Click <Help...>	Data loaded. Element defined.	Opens the Help Dialog with the BraceCalculation documentation.

**5.12.5.4 Options Pins**

Item	Action, Command	Precondition	Description
Pin Output Color	Click <Color>	Data loaded. Element defined.	Choose Output Pin color.
Pin Input Color	Click <Color>	Data loaded. Element defined.	Choose Input Pin color.
Show Pin Number 	Click Checkbox	Data loaded. Element defined.	Show the Element Pin Number in the Area.
Pin Number Font 	Popup 	Data loaded. Element defined.	Choose Pin Number Font.
Pin Number Font Height	Input Number	Data loaded. Element defined.	Choose Pin Number Font Height in pixel.
Pin Number Font Color	Click <Color>	Data loaded. Element defined.	Choose Pin Number Font color.
Show Pin Name 	Click Checkbox	Data loaded. Element defined.	Show the Element Pin Name in the Area.
Pin Name Font 	Popup 	Data loaded. Element defined.	Choose Pin Name Font.
Pin Name Font Height	Input Number	Data loaded. Element defined.	Choose Pin Name Font Height in pixel.
Pin Name Font Color	Click <Color>	Data loaded. Element defined.	Choose Pin Name Font color.
OK 	Click <OK>	Data loaded. Element defined.	Take over the set Option parameters and close the dialog.
Cancel 	Click <Cancel>	Data loaded. Element defined.	Don't take over the set Option parameters and close the dialog.
Open Documentation 	Click <Help...>	Data loaded. Element defined.	Opens the Help Dialog with the BraceCalculation documentation.

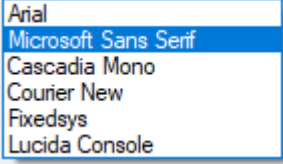


5.12.6 Curves

Item	Action, Command	Precondition	Description
New Curve 	Click <New>	Data loaded.	Create a new Curve.
Edit Curve 	Click <Edit>	Data loaded.	Open Curve dialog.
Delete Curve 	Click <Delete>	Data loaded.	Delete selected Curve from List.
Curve List 		Data loaded.	See all defined Curves.
Options Scope 	Click <Options Scope...>	Data loaded.	Open options scope dialog.

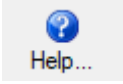


5.12.6.1 Edit Curve and Net

Item	Action, Command	Precondition	Description
Curve Name	Input Name	Data loaded. Curve defined.	Edit the Curve name. No special characters are allowed.
Curve Net Technic	Input Name	Data loaded. Curve defined.	Net technical name.
Curve Description	Input Text	Data loaded. Curve defined.	Edit the Curve description.
Wire Thick	Input thick	Data loaded. Curve defined.	Set the thickness of the wires.
Wire Color	Click <Color>	Data loaded. Curve defined.	Choose Wire color.
Curve Name Font 	Popup 	Data loaded. Curve defined.	Choose Curve Name Font.
Curve Name Font Height	Input Number	Data loaded. Curve defined.	Choose Curve Name Font Height in pixel.
Curve Name Font Color	Click <Color>	Data loaded. Curve defined.	Choose Curve Name Font color.
Connected Pin List 		Data loaded. Curve defined.	See all connected Pins.
Connected Wire List 		Data loaded. Curve defined.	See all connected Wires.



5.12.6.2 Edit Wire

Item	Action, Command	Precondition	Description
Show Wire Curve Name	Click Checkbox	Data loaded. Curve defined. Wire defined.	Selected Wire shows Curve name in the area.
Wire Curve	Click Popup	Data loaded. Curve defined. Wire defined.	Connect selected Wire to the selected Curve from the Popup menu.
Wire Area	Click Popup	Data loaded. Curve defined. Wire defined.	Set selected Wire into the selected Area from the Popup menu.
Relative Name Position X	Input Coordinate	Data loaded. Curve defined. Wire defined.	Set relative X name position from the selected Wire.
Relative Name Position Y	Input Coordinate	Data loaded. Curve defined. Wire defined.	Set relative Y name position from the selected Wire.
A Position X	Input Coordinate	Data loaded. Curve defined. Wire defined.	Set Wire side A X position. Either the X positions must be the same or the Y positions.
A Position Y	Input Coordinate	Data loaded. Curve defined. Wire defined.	Set Wire side A Y position. Either the X positions must be the same or the Y positions.
B Position X	Input Coordinate	Data loaded. Curve defined. Wire defined.	Set Wire side B X position. Either the X positions must be the same or the Y positions.
B Position Y	Input Coordinate	Data loaded. Curve defined. Wire defined.	Set Wire side B Y position. Either the X positions must be the same or the Y positions.
OK 	Click <OK>	Data loaded. Element defined.	Take over the set Option parameters and close the dialog.
Cancel 	Click <Cancel>	Data loaded. Element defined.	Don't take over the set Option parameters and close the dialog.
Open Documentation 	Click <Help...>	Data loaded. Element defined.	Opens the Help Dialog with the BraceCalculation documentation.

**5.12.6.3 Edit Axis**

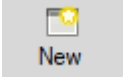
Item	Action, Command	Precondition	Description
Curve Base 	Click Popup	Data loaded. Curve defined.	Set the Curve base.

Item	Action, Command	Precondition	Description
Curve X Axis Name	Input Name	Data loaded. Curve defined.	Edit the X Axis name.
X Axis Size Name	Input Name	Data loaded. Curve defined.	Edit the X Axis Size Name
X Axis Size Name Abr.	Input Name	Data loaded. Curve defined.	Edit the X Axis Size Name abbreviation.
X Axis Unit Name	Input Name	Data loaded. Curve defined.	Edit the X Axis Unit Name
X Axis Unit Name Abr.	Input Name	Data loaded. Curve defined.	Edit the X Axis Unit Name abbreviation.
X Axis Format 	Click Popup	Data loaded. Curve defined.	Choose the number format of the X axis. Only for User base.
X Axis Start value	Input number	Data loaded. Curve defined.	Set start value of the X Axis. Only for User base.
X Axis End value	Input number	Data loaded. Curve defined.	Set end value of the X Axis. Only for User base.
X Axis Number of Steps	Input number	Data loaded. Curve defined.	Set number of steps of the X Axis. Only for User base.
X Axis Scale 	Click Popup	Data loaded. Curve defined.	Set scale of the X Axis. Only for User base.



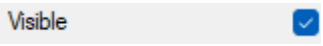
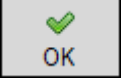
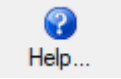
Item	Action, Command	Precondition	Description
Y Axis Size Name	Input Name	Data loaded. Curve defined.	Edit the Y Axis Size Name
Y Axis Size Name Abr.	Input Name	Data loaded. Curve defined.	Edit the Y Axis Size Name abbreviation.
Y Axis Unit Name	Input Name	Data loaded. Curve defined.	Edit the Y Axis Unit Name
Y Axis Unit Name Abr.	Input Name	Data loaded. Curve defined.	Edit the Y Axis Unit Name abbreviation.
Y Axis Format 	Click Popup	Data loaded. Curve defined.	Choose the number and display format of the Y axis.

**5.12.6.4 Edit Y Curves**

Item	Action, Command	Precondition	Description
New Y partial Curve 	Click <New>	Data loaded.	Create a new Y partial Curve.
Edit Y partial Curve 	Click <Edit>	Data loaded.	Open Y partial Curve dialog.
Delete Y partial Curve 	Click <Delete>	Data loaded.	Delete selected Y partial Curve from List.
Y partial Curve List 		Data loaded.	See all defined Y partial Curves.

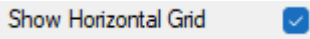
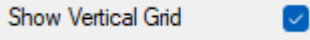
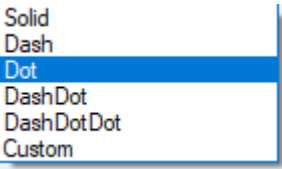
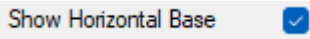
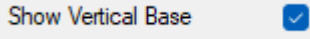
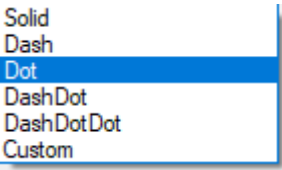


5.12.6.5 Edit Y Curve

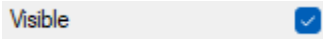
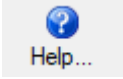
Item	Action, Command	Precondition	Description
Y Partial Curve Identification		Data loaded. Curve defined.	Selected Y partial curve identification (read only).
Y Partial Curve Name	Input Name	Data loaded. Curve defined.	Edit selected Y partial curve name.
Y Partial Curve Thickness	Input Thickness	Data loaded. Curve defined.	Set the thickness of the displayed Y partial curve.
Y Partial Curve Color	Click <Color>	Data loaded. Curve defined.	Choose color of the displayed Y partial curve.
Y Partial Curve Visibility 	Click Checkbox	Data loaded. Curve defined.	Show Y partial curve in the scope.
OK 	Click <OK>	Data loaded. Element defined.	Take over the set Option parameters and close the dialog.
Cancel 	Click <Cancel>	Data loaded. Element defined.	Don't take over the set Option parameters and close the dialog.
Open Documentation 	Click <Help...>	Data loaded. Element defined.	Opens the Help Dialog with the BraceCalculation documentation.



5.12.6.6 Options Scope

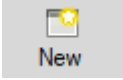
Item	Action, Command	Precondition	Description
Scope Background Color	Click >Color>	Data loaded. Curve defined.	Set the Scope background color.
Scope Diagram Background Color	Click <Color>	Data loaded. Curve defined.	Set the Scope diagram (frame) color.
Scope Show Horizontal Grid 	Click Checkbox	Data loaded. Curve defined.	Show the horizontal grid in the Scope.
Scope Show Vertical Grid 	Click Checkbox	Data loaded. Curve defined.	Show the vertical grid in the Scope.
Scope Horizontal Grid Partition	Input Number	Data loaded. Curve defined.	Set the horizontal grid partition.
Scope Vertical Grid Partition	Input Number	Data loaded. Curve defined.	Set the vertical grid partition
Scope Grid Pattern 	Click Popup	Data loaded. Curve defined.	Choose Scope grid pattern horizontal and vertical.
Scope Show Horizontal Base Grid 	Click Checkbox	Data loaded. Curve defined.	Show the horizontal base grid in the Scope.
Scope Show Vertical Base Grid 	Click Checkbox	Data loaded. Curve defined.	Show the vertical base grid in the Scope.
Scope Base Grid Pattern 	Click Popup	Data loaded. Curve defined.	Choose Scope base grid pattern horizontal and vertical.
Y Partial Curve Color	Click <Color>	Data loaded. Curve defined.	Choose color of the displayed Y partial curve.



Y Partial Curve Visibility 	Click Checkbox	Data loaded. Curve defined.	Show Y partial curve in the scope.
OK 	Click <OK>	Data loaded. Element defined.	Take over the set Option parameters and close the dialog.
Cancel 	Click <Cancel>	Data loaded. Element defined.	Don't take over the set Option parameters and close the dialog.
Open Documentation 	Click <Help...>	Data loaded. Element defined.	Opens the Help Dialog with the BraceCaclulation documentation.



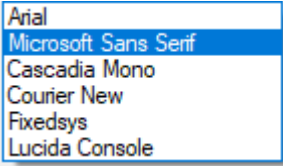
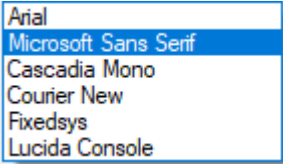
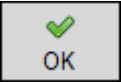
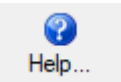
5.12.7 Text Fields

Item	Action, Command	Precondition	Description
New TextField 	Click <New>	Data loaded.	Create a new TextField.
Edit TextField 	Click <Edit>	Data loaded.	Open TextField dialog.
Delete TextField 	Click <Delete>	Data loaded.	Delete selected TextField from List.
TextField List 		Data loaded.	See all defined TextFields.

**5.12.7.1 Edit Text Field**

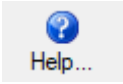
Item	Action, Command	Precondition	Description
TextField Area	Click Popup	Data loaded. TextField defined.	Choose the Area for this TextField.
TextField Title	Input Title	Data loaded. TextField defined.	Edit the TextField title.
TextField Text	Input Text	Data loaded. TextField defined.	Edit the TextField text.
TextField Height	Input Number	Data loaded. TextField defined.	Set the TextField height in drawing units.
TextField Width	Input Number	Data loaded. TextField defined.	Set the TextField width in drawing units.
TextField X Position	Input Number	Data loaded. TextField defined.	Set the TextField X position in drawing units.
TextField Y Position	Input Number	Data loaded. TextField defined.	Set the TextField Y position in drawing units.
TextField Background Color	Click <Color>	Data loaded. TextField defined.	Choose background color for this TextField.
TextField Background Transparent  Transparent	Click Checkbox	Data loaded. TextField defined.	Set the TextField background as transparent.
TextField Frame Color	Click <Color>	Data loaded. TextField defined.	Choose frame color for this TextField.
TextField Frame Transparent  Transparent	Click Checkbox	Data loaded. TextField defined.	Set the TextField frame as transparent.
TextField Text Alignment 	Click Alignment Style	Data loaded. TextField defined.	Set the TextField text alignment style. Left, Center, Right.



TextField Title Font 	Popup 	Data loaded. TextField defined.	Choose TextField title Font.
TextField Title Font Height	Input Number	Data loaded. TextField defined.	Choose TextField title font height in pixel.
TextField Title Font Color	Click <Color>	Data loaded. TextField defined.	Choose TextField title font color.
TextField Text Font 	Popup 	Data loaded. TextField defined.	Choose TextField text font.
TextField Title Font Height	Input Number	Data loaded. TextField defined.	Choose TextField text font height in pixel.
TextField Title Font Color	Click <Color>	Data loaded. TextField defined.	Choose TextField text font color.
OK 	Click <OK>	Data loaded. TextField defined.	Take over the set Area parameters and close the dialog.
Cancel 	Click <Cancel>	Data loaded. TextField defined.	Don't take over the set Area parameters and close the dialog.
Open Documentation 	Click <Help...>	Data loaded. TextField defined.	Opens the Help Dialog with the BraceCaclulation documentation.

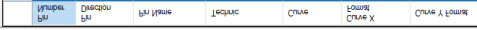


5.13 Element Dialog

Item	Action, Command	Precondition	Description
Element Selection	Click Popup	Data loaded. Element defined.	Choose the Element for detailed view.
Next Element 	Click <Next>	Data loaded. Element defined.	Choose the next Element for detailed view.
Previous Element 	Click <Previous>	Data loaded. Element defined.	Choose the previous Element for detailed view.
Close 	Click <Close>	Data loaded.	Close the Element dialog.
Open Documentation 	Click <Help...>	Data loaded.	Opens the Help Dialog with the BraceCalculation documentation.



5.13.1 Element Overview

Item	Action, Command	Precondition	Description
Element View		Data loaded. Element defined.	Show selected Element as Area View.
Element Description		Data loaded. Element defined.	Show selected Element description. Read only.
Element Pin List 		Data loaded. Element defined. Pin defined.	Show defined Element Pins with connected Curves.
Curve Settings 	Click <Curve Settings...>	Data loaded. Element defined. Curve defined.	Open Curve settings dialog.
Element Settings 	Click <Element Settings...>	Data loaded. Element defined.	Open Element settings dialog.

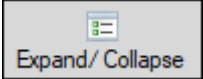
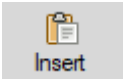
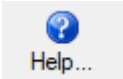


5.13.2 Element User Code

Item	Action, Command	Precondition	Description
Element User Code Window	Edit C# code	Data loaded. Element defined.	Write your own C# code for the selected Element.
Element User Code Property Window		Data loaded. Element defined. Built.	See the Element code properties. Read only.
Code Snippet 	Click <Code Snippet>	Data loaded. Element defined.	Open the code snippet dialog.

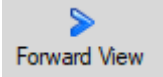
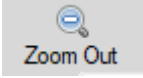


5.13.3 Code Snippet

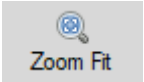
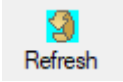
Item	Action, Command	Precondition	Description
Code Snippet Tree View	Click Snippet	Data loaded. Element defined.	Choose your code snippet.
Expand/ Collapse Tree View 	Click <Expand/ Collapse>		Expand/ collapse the code snippet tree view.
Code Snippet Window	Mark Code		See and use the code snippet.
Insert 	Click <Insert>	Data loaded. Element defined.	Insert directly the entire content of the code snippet window to the Element C# code window and close the dialog.
Cancel 	Click <Cancel>	Data loaded. Element defined.	Just close the code snippet dialog.
Open Documentation 	Click <Help...>	Data loaded. Element defined.	Opens the Help Dialog with the BraceCalculation documentation.



5.13.4 Element Scope

Item	Action, Command	Precondition	Description
Element Scope Curve Tab	Click <Tab>	Data loaded. Element defined. Built, Calculated.	See the selected and calculated curve from the selected Element.
Scope Curve Y minimum value	Input Value	Data loaded. Element defined. Built, Calculated.	Curve Y minimum scope value.
Scope Curve Y maximum value	Input Value	Data loaded. Element defined. Built, Calculated.	Curve Y maximum scope value.
Scope Curve Y middle value		Data loaded. Element defined. Built, Calculated.	Curve Y middle scope value. Read only.
Scope Curve X minimum value	Input Value	Data loaded. Element defined. Built, Calculated.	Curve X minimum scope value.
Scope Curve X maximum value	Input Value	Data loaded. Element defined. Built, Calculated.	Curve X maximum scope value.
Scope Curve X middle value		Data loaded. Element defined. Built, Calculated.	Curve Y middle scope value. Read only.
Back View 	Click <Back View>	Data loaded. Element defined. Built, Calculated.	Show scope view with last settings.
Forward View 	Click <Forward View>	Data loaded. Element defined. Built, Calculated.	Show scope view with forward settings.
Zoom In 	Click <Zoom In>	Data loaded. Element defined. Built, Calculated.	Enlarge current scope view.
Zoom Out 	Click <Zoom Out>	Data loaded. Element defined. Built, Calculated.	Reduce current scope view.



Zoom Fit 	Click <Zoom Fit>	Data loaded. Element defined. Built, Calculated.	Show calculated scope view.
Refresh 	Click <Refresh>	Data loaded. Element defined. Built, Calculated.	Refresh scope view with current values.
X Coordinate 	Mouse Cursor over scope	Data loaded. Element defined. Built, Calculated.	Show current X coordinate at mouse cursor.
Y Coordinate 	Mouse Cursor over scope	Data loaded. Element defined. Built, Calculated.	Show current Y coordinate at mouse cursor.

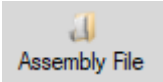


5.13.5 Element Local Event

Item	Action, Command	Precondition	Description
Element local Event Window		Data loaded. Element defined. Built, Calculated.	Show local events defined in the user code.
Clear Events 	Click <Clear Events>	Data loaded. Element defined. Built, Calculated.	Clear entire event window.



5.13.6 Element Assembly

Item	Action, Command	Precondition	Description
Element Assembly Area	Assembly defined.	Data loaded. Element defined. Assembly defined. Built, Calculated.	Show user defined Element assembly UI.
Assembly File 	Click <Assembly File>	Data loaded. Element defined. Assembly defined.	Load user assembly DLL for this Element.
Assembly File Path 	Input Path	Data loaded. Element defined. Assembly defined.	See entire assembly file path.



5.14 Engine Code Base

5.14.1 Calculation Interface

5.14.1.1 Init()

Function	<code>public void Init(Int64 maxTicks, double tickTime)</code>
Parameter	maxTicks : Number of time ticks for entire calculation tickTime : One tick time duration.
Return	-
Description	Init function at calculation start. One time call per Element. Before Loop and before Post.
Example	<pre>public void Init(Int64 maxTicks, double tickTime) { EngineTrack.WriteGlobalEvent("Init call", TrackMsgType.Information); }</pre>

5.14.1.2 Loop()

Function	<code>public void Loop(Int64 maxTicks, Int64 curTicks, double curTime, double tickTime)</code>
Parameter	maxTicks : Number of time ticks for entire calculation curTicks : Current number of ticks for current function call. curTime : Current time for current function call. tickTime : One tick time duration.
Return	-
Description	Loop function during calculation. Number of maxTicks call per Element. After Loop and before Post.
Example	<pre>public void Loop(Int64 maxTicks, Int64 curTicks, double curTime, double tickTime) { OUT_MY_CURVE.SetYDouble(curTime); }</pre>

5.14.1.3 Post()

Function	<code>public void Post(Int64 maxTicks, double tickTime)</code>
Parameter	maxTicks : Number of time ticks for entire calculation tickTime : One tick time duration.
Return	-
Description	Post function at calculation end. One time call per Element. After Loop and after Post.
Example	<pre>public void Post(Int64 maxTicks, double tickTime) { EngineTrack.WriteGlobalEvent("Post call", TrackMsgType.Information); }</pre>



5.14.2 Curve Functions

5.14.2.1 GetCurveName()

Function	<code>string GetCurveName()</code>
Parameter	-
Return	<code>string</code> : name of the curve.
Description	Get the name of the curve.
Example	<code>var myCurveName = OUT_MY_CURVE.GetCurveName();</code>

5.14.2.2 GetNumberCurves()

Function	<code>int GetNumberCurves()</code>
Parameter	-
Return	<code>int</code> : number Y curves.
Description	Get the number of partly Y curves. This number is always ≥ 1 .
Example	<code>var myYCurves = OUT_MY_CURVE.GetNumberCurves();</code>

5.14.2.3 GetXSteps()

Function	<code>Int64 GetXSteps()</code>
Parameter	-
Return	<code>Int64</code> : Number of X steps in the curve.
Description	Get the number of X steps in the curve. This number is always ≥ 10 .
Example	<code>var myCurveSteps = OUT_MY_CURVE.GetXSteps();</code>

5.14.2.4 GetXDoubleStep()

Function	<code>double GetXDoubleStep()</code>
Parameter	-
Return	<code>double</code> : step width
Description	This is the time tick from a system time curve. With a double user curve you get as step the Y-range divided by the number of steps.
Example	<code>var stepWidth = OUT_MY_CURVE.GetXDoubleStep();</code>

5.14.2.5 GetXDoubleAbsBegin()

Function	<code>double GetXDoubleAbsBegin()</code>
Parameter	-
Return	<code>double</code> : curve x array[0] value.
Description	Get the absolute first X value in the curve array. Regardless of if there is more than one sequence or when the calculation was stopped.
Example	<code>var xVal = OUT_MY_CURVE.GetXDoubleAbsBegin();</code>



5.14.2.6 GetXDoubleActBegin()

Function	double GetXDoubleActBegin()
Parameter	-
Return	double : curve x array[actual] value.
Description	Get the actual and relative X value in the curve array.
Example	<pre>var xVal = OUT_MY_CURVE.GetXDoubleActBegin();</pre>

5.14.2.7 GetXDoubleAbsEnd()

Function	double GetXDoubleAbsEnd()
Parameter	-
Return	double : curve x array[last] value.
Description	Get the absolute last X value in the curve array. Regardless of if there is more than one sequence or when the calculation was stopped.
Example	<pre>var xVal = OUT_MY_CURVE.GetXDoubleAbsEnd();</pre>

5.14.2.8 GetXDoubleActEnd()

Function	double GetXDoubleActEnd()
Parameter	-
Return	double : curve x array[actual + length] value.
Description	Get the relative actual + array length X value in the curve array.
Example	<pre>var xVal = OUT_MY_CURVE.GetXDoubleActEnd();</pre>

5.14.2.9 GetXDouble()

Function	double GetXDouble()
Parameter	-
Return	double : curve x array[actual] value
Description	Get the relative actual X value in the curve array.
Example	<pre>var xVal = OUT_MY_CURVE.GetXDouble();</pre>

5.14.2.10 GetXDoubleAbsIndex()

Function	double GetXDoubleAbsIndex(Int64 valueIndex)
Parameter	Int64 : valueIndex
Return	double : curve X array[index] value
Description	Get the absolute X value in the curve array.
Example	<pre>var xVal = OUT_MY_CURVE.GetXDoubleAbsIndex(555);</pre>

**5.14.2.11 GetXDoubleActIndex()**

Function	double GetXDoubleActIndex(Int64 valueIndex)
Parameter	Int64 : valueIndex
Return	double : curve X array[act + index] value
Description	Get the relative X value in the curve array.
Example	var xVal = OUT_MY_CURVE.GetXDoubleActIndex(-111);

5.14.2.12 GetXIntStep()

Function	Int64 GetXIntStep()
Parameter	-
Return	Int64 : step width
Description	This is 1 from a system tick curve. With a Int64 user curve you get as step the Y-range divided by the number of steps.
Example	var stepWidth = OUT_MY_CURVE.GetXIntStep();

5.14.2.13 GetXIntAbsBegin()

Function	Int64 GetXIntAbsBegin()
Parameter	-
Return	Int64 : curve x array[0] value.
Description	Get the absolute first X value in the curve array. Regardless of if there is more than one sequence or when the calculation was stopped.
Example	var xVal = OUT_MY_CURVE.GetXIntAbsBegin();

5.14.2.14 GetXIntActBegin()

Function	Int64 GetXIntActBegin()
Parameter	-
Return	Int64 : curve x array[actual] value.
Description	Get the actual and relative X value in the curve array.
Example	var xVal = OUT_MY_CURVE.GetXIntActBegin();

5.14.2.15 GetXIntAbsEnd()

Function	Int64 GetXDoubleAbsEnd()
Parameter	-
Return	Int64 : curve x array[last] value.
Description	Get the absolute last X value in the curve array. Regardless of if there is more than one sequence or when the calculation was stopped.
Example	var xVal = OUT_MY_CURVE.GetXIntAbsEnd();

**5.14.2.16 GetXIntActEnd()**

Function	Int64 GetXDoubleActEnd()
Parameter	-
Return	Int64 : curve x array[actual + length] value.
Description	Get the relative actual + array length X value in the curve array.
Example	<pre>var xVal = OUT_MY_CURVE.GetXIntActEnd();</pre>

5.14.2.17 GetXInt()

Function	Int64 GetXInt()
Parameter	-
Return	Int64 : curve x array[actual] value
Description	Get the relative actual X value in the curve array.
Example	<pre>var xVal = OUT_MY_CURVE.GetXInt();</pre>

5.14.2.18 GetXIntAbsIndex()

Function	Int64 GetXIntAbsIndex(Int64 valueIndex)
Parameter	Int64 : valueIndex
Return	Int64 : curve X array[index] value
Description	Get the absolute X value in the curve array.
Example	<pre>var xVal = OUT_MY_CURVE.GetXIntAbsIndex(555);</pre>

5.14.2.19 GetXIntActIndex()

Function	Int64 GetXIntActIndex(Int64 valueIndex)
Parameter	Int64 : valueIndex
Return	Int64 : curve X array[act + index] value
Description	Get the relative X value in the curve array.
Example	<pre>var xVal = OUT_MY_CURVE.GetXIntActIndex(-111);</pre>



5.14.2.20 GetYDoubleFromXDouble

Function	double GetYDoubleFromXDouble(double x)
Parameter	double : curve X value
Return	double : curve Y value
Description	Get the Y curve[0] value according to the X curve value.
Example	var yVal = OUT_MY_CURVE.GetYDoubleFromXDouble(123.456);

Function	double GetYDoubleFromXDouble(int curveIndex, double x)
Parameter	int : Y curve index double : curve X value
Return	double : curve Y value
Description	Get the Y curve[curveIndex] value according to the X curve value.
Example	var yVal = OUT_MY_CURVE.GetYDoubleFromXDouble(4, 123.456);

5.14.2.21 GetYDoubleFromXInt()

Function	double GetYDoubleFromXInt(Int64 x)
Parameter	Int64 : curve X value
Return	double : curve Y value
Description	Get the Y curve[0] value according to the X curve value.
Example	var yVal = OUT_MY_CURVE.GetYDoubleFromXInt(123456);

Function	double GetYDoubleFromXInt(int curveIndex, Int64 x)
Parameter	int : Y curve index Int64 : curve X value
Return	double : curve Y value
Description	Get the Y curve[curveIndex] value according to the X curve value.
Example	var yVal = OUT_MY_CURVE.GetYDoubleFromXDouble(4, 123456);

5.14.2.22 GetYDouble()

Function	double GetYDouble()
Parameter	-
Return	double : curve Y value
Description	Get the actual Y curve[0] value according to the current calculation time (tick). For system time and system tick curves.
Example	var yVal = OUT_MY_CURVE.GetYDouble();

Function	double GetYDouble(int curveIndex)
Parameter	int : Y curve index
Return	double : curve Y value
Description	Get the actual Y curve[curveIndex] value according to the current calculation time (tick). For system time and system tick curves.
Example	var yVal = OUT_MY_CURVE.GetYDouble(4);

**5.14.2.23 GetYDoublePrev()**

Function	double GetYDoublePrev()
Parameter	-
Return	double : curve Y value
Description	Get the previous (actual-1) Y curve[0] value according to the current calculation time (tick). For system time and system tick curves.
Example	<pre>var yVal = OUT_MY_CURVE.GetYDoublePrev();</pre>

Function	double GetYDoublePrev(int curveIndex)
Parameter	int : Y curve index
Return	double : curve Y value
Description	Get the previous (actual-1) Y curve[curveIndex] value according to the current calculation time (tick). For system time and system tick curves.
Example	<pre>var yVal = OUT_MY_CURVE.GetYDoublePrev(3);</pre>



5.14.2.24 GetYDoubleAbsIndex()

Function	double GetYDoubleAbsIndex(Int64 index)
Parameter	Int64 : absolute curve value index
Return	double : curve Y value
Description	Get the curve[0] Y value at the absolute index position.
Example	var yVal = OUT_MY_CURVE.GetYDoubleAbsIndex(12345);

Function	double GetYDoubleAbsIndex(int curveIndex, Int64 index)
Parameter	int : curve index Int64 : absolute curve value index
Return	double : curve Y value
Description	Get the curve[curveIndex] Y value at the absolute index position.
Example	var yVal = OUT_MY_CURVE.GetYDoubleAbsIndex(3, 12345);

5.14.2.25 GetYDoubleActIndex()

Function	double GetYDoubleActIndex(Int64 index)
Parameter	Int64 : relative curve value index
Return	double : curve Y value
Description	Get the curve[0] Y value at the relative actual+index position.
Example	var yVal = OUT_MY_CURVE.GetYDoubleActIndex(-5);

Function	double GetYDoubleActIndex(int curveIndex, Int64 index)
Parameter	int : curve index Int64 : relative curve value index
Return	double : curve Y value
Description	Get the curve[curveIndex] Y value at the relative actual+index position.
Example	var yVal = OUT_MY_CURVE.GetYDoubleActIndex(3, -5);

5.14.2.26 SetYDouble()

Function	void SetYDouble(double y)
Parameter	double : curve Y value
Return	-
Description	Set the actual Y curve[0] value according to the current calculation time (tick). For system time and system tick curves.
Example	OUT_MY_CURVE.SetYDouble(0.876);

Function	void SetYDouble(int curveIndex, double y)
Parameter	int : curve index double : curve Y value
Return	-
Description	Set the actual Y curve[curveIndex] value according to the current calculation time (tick). For system time and system tick curves.
Example	OUT_MY_CURVE.SetYDouble(4, 0.876);

**5.14.2.27 SetYDoubleAbsIndex()**

Function	void SetYDoubleAbsIndex(Int64 valueIndex, double y)
Parameter	Int64 : curve Y value index double : curve Y value
Return	-
Description	Set the absolute Y curve[0] value at the index position.
Example	OUT_MY_CURVE.SetYDoubleAbsIndex(1234, 0.876);

Function	void SetYDoubleAbsIndex(int curveIndex, Int64 valueIndex, double y)
Parameter	int : curve index Int64 : curve Y value index double : curve Y value
Return	-
Description	Set the absolute Y curve[curveIndex] value at the index position.
Example	OUT_MY_CURVE.SetYDoubleAbsIndex(4, 1234, 0.876);



5.14.2.28 GetYIntFromXDouble

Function	Int64 GetYIntFromXDouble(double x)
Parameter	double : curve X value
Return	Int64 : curve Y value
Description	Get the Y curve[0] value according to the X curve value.
Example	<pre>var yVal = OUT_MY_CURVE.GetYIntFromXDouble(123.456);</pre>

Function	Int64 GetYIntFromXDouble(int curveIndex, double x)
Parameter	int : Y curve index double : curve X value
Return	Int64 : curve Y value
Description	Get the Y curve[curveIndex] value according to the X curve value.
Example	<pre>var yVal = OUT_MY_CURVE.GetYIntFromXDouble(4, 123.456);</pre>

5.14.2.29 GetYIntFromXInt()

Function	Int64 GetYIntFromXInt(Int64 x)
Parameter	Int64 : curve X value
Return	Int64 : curve Y value
Description	Get the Y curve[0] value according to the X curve value.
Example	<pre>var yVal = OUT_MY_CURVE.GetYIntFromXInt(123456);</pre>

Function	Int64 GetYIntFromXInt(int curveIndex, Int64 x)
Parameter	int : Y curve index Int64 : curve X value
Return	Int64 : curve Y value
Description	Get the Y curve[curveIndex] value according to the X curve value.
Example	<pre>var yVal = OUT_MY_CURVE.GetYIntFromXInt(4, 123456);</pre>

5.14.2.30 GetYInt()

Function	Int64 GetYInt()
Parameter	-
Return	Int64 : curve Y value
Description	Get the actual Y curve[0] value according to the current calculation time (tick). For system time and system tick curves.
Example	<pre>var yVal = OUT_MY_CURVE.GetYInt();</pre>

Function	Int64 GetYInt(int curveIndex)
Parameter	int : Y curve index
Return	Int64 : curve Y value
Description	Get the actual Y curve[curveIndex] value according to the current calculation time (tick). For system time and system tick curves.
Example	<pre>var yVal = OUT_MY_CURVE.GetYInt(4);</pre>

**5.14.2.31 GetYIntPrev()**

Function	Int64 GetYIntPrev()
Parameter	-
Return	Int64 : curve Y value
Description	Get the previous (actual-1) Y curve[0] value according to the current calculation time (tick). For system time and system tick curves.
Example	<pre>var yVal = OUT_MY_CURVE.GetYIntPrev();</pre>

Function	Int64 GetYIntPrev(int curveIndex)
Parameter	int : Y curve index
Return	Int64 : curve Y value
Description	Get the previous (actual-1) Y curve[curveIndex] value according to the current calculation time (tick). For system time and system tick curves.
Example	<pre>var yVal = OUT_MY_CURVE.GetYIntPrev(4);</pre>



5.14.2.32 GetYIntAbsIndex()

Function	Int64 GetYIntAbsIndex(Int64 index)
Parameter	Int64 : absolute curve value index
Return	Int64 : curve Y value
Description	Get the curve[0] Y value at the absolute index position.
Example	<pre>var yVal = OUT_MY_CURVE.GetYIntAbsIndex(12345);</pre>

Function	Int64 GetYIntAbsIndex(int curveIndex, Int64 index)
Parameter	int : curve index Int64 : absolute curve value index
Return	Int64 : curve Y value
Description	Get the curve[curveIndex] Y value at the absolute index position.
Example	<pre>var yVal = OUT_MY_CURVE.GetYIntAbsIndex(3, 12345);</pre>

5.14.2.33 GetYIntActIndex()

Function	Int64 GetYIntActIndex(Int64 index)
Parameter	Int64 : relative curve value index
Return	Int64 : curve Y value
Description	Get the curve[0] Y value at the relative actual+index position.
Example	<pre>var yVal = OUT_MY_CURVE.GetYIntActIndex(-5);</pre>

Function	Int64 GetYIntActIndex(int curveIndex, Int64 index)
Parameter	int : curve index Int64 : relative curve value index
Return	Int64 : curve Y value
Description	Get the curve[curveIndex] Y value at the relative actual+index position.
Example	<pre>var yVal = OUT_MY_CURVE.GetYIntActIndex(3, -5);</pre>

5.14.2.34 SetYInt()

Function	void SetYInt(Int64 y)
Parameter	Int64 : curve Y value
Return	-
Description	Set the actual Y curve[0] value according to the current calculation time (tick). For system time and system tick curves.
Example	<pre>OUT_MY_CURVE.SetYInt(876);</pre>

Function	void SetYInt(int curveIndex, Int64 y)
Parameter	int : curve index Int64 : curve Y value
Return	-
Description	Set the actual Y curve[curveIndex] value according to the current calculation time (tick). For system time and system tick curves.
Example	<pre>OUT_MY_CURVE.SetYInt(4, 876);</pre>

**5.14.2.35 SetYIntAbsIndex()**

Function	void SetYIntAbsIndex(Int64 valueIndex, Int64 y)
Parameter	Int64 : curve Y value index Int64 : curve Y value
Return	-
Description	Set the absolute Y curve[0] value at the index position.
Example	OUT_MY_CURVE.SetYIntAbsIndex(1234, 876);

Function	void SetYIntAbsIndex(int curveIndex, Int64 valueIndex, Int64 y)
Parameter	int : curve index Int64 : curve Y value index Int64 : curve Y value
Return	-
Description	Set the absolute Y curve[curveIndex] value at the index position.
Example	OUT_MY_CURVE.SetYIntAbsIndex(4, 1234, 876);



5.14.2.36 GetYByteArrayFromXDouble

Function	byte[] GetYByteArrayFromXDouble(double x)
Parameter	double : curve X value
Return	byte[] : curve Y value
Description	Get the Y curve[0] value according to the X curve value.
Example	var yVal = OUT_MY_CURVE.GetYByteArrayFromXDouble(123.456);

Function	byte[] GetYByteArrayFromXDouble(int curveIndex, double x)
Parameter	int : Y curve index double : curve X value
Return	byte[] : curve Y value
Description	Get the Y curve[curveIndex] value according to the X curve value.
Example	var yVal = OUT_MY_CURVE.GetYByteArrayFromXDouble(4, 123.456);

5.14.2.37 GetYByteArrayFromXInt()

Function	byte[] GetYByteArrayFromXInt(Int64 x)
Parameter	Int64 : curve X value
Return	byte[] : curve Y value
Description	Get the Y curve[0] value according to the X curve value.
Example	var yVal = OUT_MY_CURVE.GetYByteArrayFromXInt(123456);

Function	byte[] GetYByteArrayFromXInt(int curveIndex, Int64 x)
Parameter	int : Y curve index Int64 : curve X value
Return	byte[] : curve Y value
Description	Get the Y curve[curveIndex] value according to the X curve value.
Example	var yVal = OUT_MY_CURVE.GetYByteArrayFromXInt(4, 123456);

5.14.2.38 GetYByteArray()

Function	byte[] GetYByteArray()
Parameter	-
Return	byte[] : curve Y value
Description	Get the actual Y curve[0] value according to the current calculation time (tick). For system time and system tick curves.
Example	var yVal = OUT_MY_CURVE.GetYByteArray();

Function	byte[] GetYByteArray(int curveIndex)
Parameter	int : Y curve index
Return	byte[] : curve Y value
Description	Get the actual Y curve[curveIndex] value according to the current calculation time (tick). For system time and system tick curves.
Example	var yVal = OUT_MY_CURVE.GetYByteArray(4);

**5.14.2.39 GetYByteArrayPrev()**

Function	<code>byte[] GetYByteArrayPrev()</code>
Parameter	-
Return	<code>byte[]</code> : curve Y value
Description	Get the previous (actual-1) Y curve[0] value according to the current calculation time (tick). For system time and system tick curves.
Example	<code>var yVal = OUT_MY_CURVE.GetYByteArrayPrev();</code>

Function	<code>byte[] GetYByteArrayPrev(int curveIndex)</code>
Parameter	<code>int</code> : Y curve index
Return	<code>byte[]</code> : curve Y value
Description	Get the previous (actual-1) Y curve[curveIndex] value according to the current calculation time (tick). For system time and system tick curves.
Example	<code>var yVal = OUT_MY_CURVE.GetYByteArrayPrev(4);</code>



5.14.2.40 GetYByteArrayAbsIndex()

Function	byte[] GetYByteArrayAbsIndex(Int64 index)
Parameter	Int64 : absolute curve value index
Return	byte[] : curve Y value
Description	Get the curve[0] Y value at the absolute index position.
Example	var yVal = OUT_MY_CURVE.GetYByteArrayAbsIndex(12345);

Function	byte[] GetYIntAbsIndex(int curveIndex, Int64 index)
Parameter	int : curve index Int64 : absolute curve value index
Return	byte[] : curve Y value
Description	Get the curve[curveIndex] Y value at the absolute index position.
Example	var yVal = OUT_MY_CURVE.GetYByteArrayAbsIndex(3, 12345);

5.14.2.41 GetYByteArrayActIndex()

Function	byte[] GetYByteArrayActIndex(Int64 index)
Parameter	Int64 : relative curve value index
Return	byte[] : curve Y value
Description	Get the curve[0] Y value at the relative actual+index position.
Example	var yVal = OUT_MY_CURVE.GetYByteArrayActIndex(-5);

Function	byte[] GetYByteArrayActIndex(int curveIndex, Int64 index)
Parameter	int : curve index Int64 : relative curve value index
Return	byte[] : curve Y value
Description	Get the curve[curveIndex] Y value at the relative actual+index position.
Example	var yVal = OUT_MY_CURVE.GetYByteArrayActIndex(3, -5);

5.14.2.42 SetYByteArray()

Function	void SetYByteArray(byte[] y)
Parameter	byte[] : curve Y value
Return	-
Description	Set the actual Y curve[0] value according to the current calculation time (tick). For system time and system tick curves.
Example	OUT_MY_CURVE.SetYByteArray(new byte[] {0x55, 0xAA});

Function	void SetYByteArray(int curveIndex, byte[] y)
Parameter	int : curve index byte[] : curve Y value
Return	-
Description	Set the actual Y curve[curveIndex] value according to the current calculation time (tick). For system time and system tick curves.
Example	OUT_MY_CURVE.SetYByteArray(4, new byte[] {0x55, 0xAA});

**5.14.2.43 SetYByteArrayAbsIndex()**

Function	void SetYByteArrayAbsIndex(Int64 valueIndex, byte[] y)
Parameter	Int64 : curve Y value index byte[] : curve Y value
Return	-
Description	Set the absolute Y curve[0] value at the index position.
Example	OUT_MY_CURVE.SetYByteArrayAbsIndex(1234, new byte[]{0x55, 0xAA});

Function	void SetYByteArrayAbsIndex(int curveIndex, Int64 valueIndex, byte[] y)
Parameter	int : curve index Int64 : curve Y value index byte[] : curve Y value
Return	-
Description	Set the absolute Y curve[curveIndex] value at the index position.
Example	OUT_MY_CURVE.SetYByteArrayAbsIndex(4, 1234, new byte[]{0x55, 0xAA});



5.15 Engine Library

5.15.1 Analog Logic

5.15.1.1 CreateAnalogLogic()

Function	<code>int CreateAnalogLogic(double lowThresholdInput, double highThresholdInput, double lowLevelOutput, double highLevelOutput)</code>
Parameter	<code>double lowThresholdInput</code> : low threshold input value to recognize logic low. <code>double highThresholdInput</code> : high threshold input value to recognize logic high. <code>double lowLevelOutput</code> : low level output value for logic low. <code>double highLevelOutput</code> : high level output value for logic high.
Return	<code>int</code> : object key for AnalogLogic.
Description	Create an AnalogLogic object for input and output curves.
Example	<pre>var aKey = EngineLibrary.AnalogLogic.CreateAnalogLogic(0.8, 2.0, -5.0, 5.0);</pre>

5.15.1.2 CreateAnalogStream()

Function	<code>int CreateAnalogStream(byte[] data, Int64 delayTicks, Int64 periodTicks, Int64 repeate, double lowLevelOutput, double highLevelOutput)</code>
Parameter	<code>byte[] data</code> : stream data buffer in bytes. <code>Int64 delayTicks</code> : delay in ticks after calculation start. <code>Int64 periodTicks</code> : signal period in ticks sets the frequency of the logic signal. <code>Int64 repeat</code> : number of data buffer stream repeats in the signal.
Return	<code>int</code> : object key for AnalogStream.
Description	Create an AnalogStream object for output curves.
Example	<pre>var aKey = EngineLibrary.AnalogLogic.CreateAnalogStream(new byte[]{0x55, 0xAA}, 1000, 10, 20, -5.0, 5.0);</pre>



5.15.1.3 CurveBUF()

Function	<code>void CurveBUF(int key, IEngineCurve inCurve, IEngineCurve outCurve)</code>						
Parameter	<code>int key</code> : object key from <code>CreateAnalogLogic()</code> . <code>IEngineCurve inCurve</code> : input curve. <code>IEngineCurve outCurve</code> : output curve.						
Return	-						
Description	Set the output levels in the output curve[0] according to the input curve[0]. The output curve logic is equal to the input curve logic. <table><tr><td><code>inCurve</code></td><td><code>outCurve</code></td></tr><tr><td>1</td><td>1</td></tr><tr><td>0</td><td>0</td></tr></table> The entire output curve[0] is calculated at once. Use in <code>Init()</code> or <code>Post()</code> functions.	<code>inCurve</code>	<code>outCurve</code>	1	1	0	0
<code>inCurve</code>	<code>outCurve</code>						
1	1						
0	0						
Example	<code>EngineLibrary.AnalogLogic.CurveBUF(aKey, IN_MY_CURVE, OUT_MY_CURVE);</code>						

Function	<code>void CurveBUF(int key, IEngineCurve inCurve, int inCurveIndex, IEngineCurve outCurve, int outCurveIndex)</code>						
Parameter	<code>int key</code> : object key from <code>CreateAnalogLogic()</code> . <code>IEngineCurve inCurve</code> : input curve. <code>int inCurveIndex</code> : input curve index. <code>IEngineCurve outCurve</code> : output curve. <code>int outCurveIndex</code> : output curve index.						
Return	-						
Description	Set the output levels in the output curve[outCurveIndex] according to the input curve[inCurveIndex]. The output curve logic is equal to the input curve logic. <table><tr><td><code>inCurve</code></td><td><code>outCurve</code></td></tr><tr><td>1</td><td>1</td></tr><tr><td>0</td><td>0</td></tr></table> The entire output curve[outCurveIndex] is calculated at once. Use in <code>Init()</code> or <code>Post()</code> functions.	<code>inCurve</code>	<code>outCurve</code>	1	1	0	0
<code>inCurve</code>	<code>outCurve</code>						
1	1						
0	0						
Example	<code>EngineLibrary.AnalogLogic.CurveBUF(aKey, IN_MY_CURVE, 3, OUT_MY_CURVE, 3);</code>						



5.15.1.4 CurveNOT()

Function	<pre>void CurveNOT(int key, IEngineCurve inCurve, IEngineCurve outCurve)</pre>						
Parameter	<pre>int key : object key from CreateAnalogLogic(). IEngineCurve inCurve : input curve. IEngineCurve outCurve : output curve.</pre>						
Return	-						
Description	Set the output levels in the output curve[0] according to the input curve[0]. The output curve logic is inverted to the input curve logic. <table><tr><td>inCurve</td><td>outCurve</td></tr><tr><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td></tr></table> The entire output curve[0] is calculated at once. Use in Init() or Post() functions.	inCurve	outCurve	1	0	0	1
inCurve	outCurve						
1	0						
0	1						
Example	<pre>EngineLibrary.AnalogLogic.CurveNOT(aKey, IN_MY_CURVE, OUT_MY_CURVE);</pre>						

Function	<pre>void CurveNOT(int key, IEngineCurve inCurve, int inCurveIndex, IEngineCurve outCurve, int outCurveIndex)</pre>						
Parameter	<pre>int key : object key from CreateAnalogLogic(). IEngineCurve inCurve : input curve. int inCurveIndex : input curve index. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.</pre>						
Return	-						
Description	Set the output levels in the output curve[outCurveIndex] according to the input curve[inCurveIndex]. The output curve logic is inverted to the input curve logic. <table><tr><td>inCurve</td><td>outCurve</td></tr><tr><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td></tr></table> The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.	inCurve	outCurve	1	0	0	1
inCurve	outCurve						
1	0						
0	1						
Example	<pre>EngineLibrary.AnalogLogic.CurveNOT(aKey, IN_MY_CURVE, 3, OUT_MY_CURVE, 3);</pre>						



5.15.1.5 CurveOR()

Function	void CurveOR(int key, IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)
Parameter	int key : object key from CreateAnalogLogic(). IEngineCurve inCurveA : first input curve. IEngineCurve inCurveB : second input curve. IEngineCurve outCurve : output curve.
Return	-
Description	Set the output levels in the output curve[0] as a logic OR combination from input curve[0] A and input curve[0] B. incurveA inCurveB outCurve 0 0 0 1 0 1 0 1 1 1 1 1 The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	EngineLibrary.AnalogLogic.CurveOR(aKey, IN_MY_CURVE_A, IN_MY_CURVE_B, OUT_MY_CURVE);

Function	void CurveOR(int key, IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)
Parameter	int key : object key from CreateAnalogLogic(). IEngineCurve inCurveA : first input curve. int inCurveIndexA : first input curve index. IEngineCurve inCurveB : second input curve. int inCurveIndexB : second input curve index. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.
Return	-
Description	Set the output levels in the output curve[outCurveIndex] as a logic OR combination between input curve[inCurveIndexA] A and input curve[inCurveIndexB] B. incurveA inCurveB outCurve 0 0 0 1 0 1 0 1 1 1 1 1 The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	EngineLibrary.AnalogLogic.CurveOR(aKey, IN_MY_CURVE_A, 2, IN_MY_CURVE_B, 3, OUT_MY_CURVE, 0);



5.15.1.6 CurveNOR()

Function	<pre>void CurveNOR(int key, IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)</pre>															
Parameter	<pre>int key : object key from CreateAnalogLogic(). IEngineCurve inCurveA : first input curve. IEngineCurve inCurveB : second input curve. IEngineCurve outCurve : output curve.</pre>															
Return	-															
Description	Set the output levels in the output curve[0] as a logic NOR combination from input curve[0] A and input curve[0] B. <table><tr><td>incurveA</td><td>inCurveB</td><td>outCurve</td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table> The entire output curve[0] is calculated at once. Use in Init() or Post() functions.	incurveA	inCurveB	outCurve	0	0	1	1	0	0	0	1	0	1	1	0
incurveA	inCurveB	outCurve														
0	0	1														
1	0	0														
0	1	0														
1	1	0														
Example	<pre>EngineLibrary.AnalogLogic.CurveNOR(aKey, IN_MY_CURVE_A, IN_MY_CURVE_B, OUT_MY_CURVE);</pre>															

Function	<pre>void CurveNOR(int key, IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)</pre>															
Parameter	<pre>int key : object key from CreateAnalogLogic(). IEngineCurve inCurveA : first input curve. int inCurveIndexA : first input curve index. IEngineCurve inCurveB : second input curve. int inCurveIndexB : second input curve index. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.</pre>															
Return	-															
Description	Set the output levels in the output curve[outCurveIndex] as a logic NOR combination between input curve[inCurveIndexA] A and input curve[inCurveIndexB] B. <table><tr><td>incurveA</td><td>inCurveB</td><td>outCurve</td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table> The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.	incurveA	inCurveB	outCurve	0	0	1	1	0	0	0	1	0	1	1	0
incurveA	inCurveB	outCurve														
0	0	1														
1	0	0														
0	1	0														
1	1	0														
Example	<pre>EngineLibrary.AnalogLogic.CurveNOR(aKey, IN_MY_CURVE_A, 2, IN_MY_CURVE_B, 3 OUT_MY_CURVE, 0);</pre>															



5.15.1.7 CurveXOR()

Function	<pre>void CurveXOR(int key, IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)</pre>
Parameter	<pre>int key : object key from CreateAnalogLogic(). IEngineCurve inCurveA : first input curve. IEngineCurve inCurveB : second input curve. IEngineCurve outCurve : output curve.</pre>
Return	-
Description	Set the output levels in the output curve[0] as a logic XOR combination from input curve[0] A and input curve[0] B. <pre>incurveA inCurveB outCurve 0 0 0 1 0 1 0 1 1 1 1 0</pre> The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	<pre>EngineLibrary.AnalogLogic.CurveXOR(aKey, IN_MY_CURVE_A, IN_MY_CURVE_B, OUT_MY_CURVE);</pre>

Function	<pre>void CurveXOR(int key, IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)</pre>															
Parameter	<pre>int key : object key from CreateAnalogLogic(). IEngineCurve inCurveA : first input curve. int inCurveIndexA : first input curve index. IEngineCurve inCurveB : second input curve. int inCurveIndexB : second input curve index. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.</pre>															
Return	-															
Description	Set the output levels in the output curve[outCurveIndex] as a logic XOR combination between input curve[inCurveIndexA] A and input curve[inCurveIndexB] B. <table><tr><td>incurveA</td><td>inCurveB</td><td>outCurve</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table> The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.	incurveA	inCurveB	outCurve	0	0	0	1	0	1	0	1	1	1	1	0
incurveA	inCurveB	outCurve														
0	0	0														
1	0	1														
0	1	1														
1	1	0														
Example	<pre>EngineLibrary.AnalogLogic.CurveXOR(aKey, IN_MY_CURVE_A, 2, IN_MY_CURVE_B, 3, OUT_MY_CURVE, 0);</pre>															



5.15.1.8 CurveAND()

Function	void CurveAND(int key, IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)
Parameter	int key : object key from CreateAnalogLogic(). IEngineCurve inCurveA : first input curve. IEngineCurve inCurveB : second input curve. IEngineCurve outCurve : output curve.
Return	-
Description	Set the output levels in the output curve[0] as a logic AND combination from input curve[0] A and input curve[0] B. incurveA inCurveB outCurve 0 0 0 1 0 0 0 1 0 1 1 1 The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	EngineLibrary.AnalogLogic.CurveAND(aKey, IN_MY_CURVE_A, IN_MY_CURVE_B, OUT_MY_CURVE);

Function	void CurveAND(int key, IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)
Parameter	int key : object key from CreateAnalogLogic(). IEngineCurve inCurveA : first input curve. int inCurveIndexA : first input curve index. IEngineCurve inCurveB : second input curve. int inCurveIndexB : second input curve index. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.
Return	-
Description	Set the output levels in the output curve[outCurveIndex] as a logic AND combination between input curve[inCurveIndexA] A and input curve[inCurveIndexB] B. incurveA inCurveB outCurve 0 0 0 1 0 0 0 1 0 1 1 1 The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	EngineLibrary.AnalogLogic.CurveAND(aKey, IN_MY_CURVE_A, 2, IN_MY_CURVE_B, 3 OUT_MY_CURVE, 0);



5.15.1.9 CurveNAND()

Function	void CurveNAND(int key, IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)
Parameter	int key : object key from CreateAnalogLogic(). IEngineCurve inCurveA : first input curve. IEngineCurve inCurveB : second input curve. IEngineCurve outCurve : output curve.
Return	-
Description	Set the output levels in the output curve[0] as a logic NAND combination from input curve[0] A and input curve[0] B. incurveA inCurveB outCurve 0 0 1 1 0 1 0 1 1 1 1 0 The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	EngineLibrary.AnalogLogic.CurveNAND(aKey, IN_MY_CURVE_A, IN_MY_CURVE_B, OUT_MY_CURVE);

Function	void CurveNAND(int key, IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)
Parameter	int key : object key from CreateAnalogLogic(). IEngineCurve inCurveA : first input curve. int inCurveIndexA : first input curve index. IEngineCurve inCurveB : second input curve. int inCurveIndexB : second input curve index. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.
Return	-
Description	Set the output levels in the output curve[outCurveIndex] as a logic NAND combination between input curve[inCurveIndexA] A and input curve[inCurveIndexB] B. incurveA inCurveB outCurve 0 0 1 1 0 1 0 1 1 1 1 0 The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	EngineLibrary.AnalogLogic.CurveNAND(aKey, IN_MY_CURVE_A, 2, IN_MY_CURVE_B, 3, OUT_MY_CURVE, 0);



5.15.1.10 CurveStream()

Function	<code>void CurveStream(int key, bool logic, bool msbFirst, IEngineCurve outCurve)</code>
Parameter	<code>int key</code> : object key from CreateAnalogLogic(). <code>bool logic</code> : True - 1 is high, 0 is low. False - 1 is low, 0 is high. <code>bool msbFirst</code> : True - MSB byte first. False - LSB byte first. <code>IEngineCurve outCurve</code> : output curve.
Return	-
Description	Set a created stream object to a curve[0]. The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.AnalogLogic.CurveStream(aKey, true, false, OUT_MY_CURVE);</code>

Function	<code>void CurveStream(int key, bool logic, bool msbFirst, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>int key</code> : object key from CreateAnalogLogic(). <code>bool logic</code> : True - 1 is high, 0 is low. False - 1 is low, 0 is high. <code>bool msbFirst</code> : True - MSB byte first. False - LSB byte first. <code>IEngineCurve outCurve</code> : output curve. <code>int outCurveIndex</code> : output curve index.
Return	-
Description	Set a created stream object to a curve[outCurveIndex]. The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.AnalogLogic.CurveStream(aKey, true, false, OUT_MY_CURVE, 7);</code>

**5.15.1.11 ActBuf()**

Function	<code>double ActBUF(int key, IEngineCurve inCurve, IEngineCurve outCurve)</code>
Parameter	<code>int key</code> : object key from <code>CreateAnalogLogic()</code> . <code>IEngineCurve inCurve</code> : input curve. <code>IEngineCurve outCurve</code> : output curve.
Return	<code>double</code> : new calculated actual Y-value.
Description	Set the output level in the actual output curve[0] Y-value according to the actual input curve[0] Y-value. The output curve logic is equal to the input curve logic. <pre>inCurve outCurve 1 1 0 0</pre> Only the actual output curve[0] Y-value is calculated. Use in <code>Loop()</code> function for system curves.
Example	<pre>var yVal = EngineLibrary.AnalogLogic.ActBUF(aKey, IN_MY_CURVE, OUT_MY_CURVE);</pre>

Function	<code>double ActBUF(int key, IEngineCurve inCurve, int inCurveIndex, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>int key</code> : object key from <code>CreateAnalogLogic()</code> . <code>IEngineCurve inCurve</code> : input curve. <code>int inCurveIndex</code> : input curve index. <code>IEngineCurve outCurve</code> : output curve. <code>int outCurveIndex</code> : output curve index.
Return	<code>double</code> : new calculated actual Y-value.
Description	Set the output level in the actual output curve[outCurveIndex] Y-value according to the actual input curve[inCurveIndex] Y-value. The output curve logic is equal to the input curve logic. <pre>inCurve outCurve 1 1 0 0</pre> Only the actual output curve[outCurveIndex] Y-value is calculated. Use in <code>Loop()</code> function for system curves.
Example	<pre>var yVal = EngineLibrary.AnalogLogic.ActBUF(aKey, IN_MY_CURVE, 3, OUT_MY_CURVE, 3);</pre>

**5.15.1.12 ActNOT()**

Function	<code>doubl eActNOT(int key, IEngineCurve inCurve, IEngineCurve outCurve)</code>
Parameter	<code>int key : object key from CreateAnalogLogic(). IEngineCurve inCurve : input curve. IEngineCurve outcurve : output curve.</code>
Return	<code>double : new calculated actual Y-value.</code>
Description	Set the output level in the actual output curve[0] Y-value according to the actual input curve[0] Y-value. The output curve logic is inverted to the input curve logic. <pre>inCurve outCurve 1 0 0 1</pre> Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves.
Example	<pre>var yVal = EngineLibrary.AnalogLogic.ActNOT(aKey, IN_MY_CURVE, OUT_MY_CURVE);</pre>

Function	<code>double ActNOT(int key, IEngineCurve inCurve, int inCurveIndex, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>int key : object key from CreateAnalogLogic(). IEngineCurve inCurve : input curve. int inCurveIndex : input curve index. IEngineCurve outcurve : output curve. int outCurveIndex : output curve index.</code>
Return	<code>double : new calculated actual Y-value.</code>
Description	Set the output level in the actual output curve[outCurveIndex] Y-value according to the actual input curve[inCurveIndex] Y-value. The output curve logic is inverted to the input curve logic. <pre>inCurve outCurve 1 0 0 1</pre> Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves.
Example	<pre>var yVal = EngineLibrary.AnalogLogic.ActNOT(aKey, IN_MY_CURVE, 3, OUT_MY_CURVE, 3);</pre>



5.15.1.13 ActOR()

Function	double ActOR(int key, IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)
Parameter	int key : object key from CreateAnalogLogic(). IEngineCurve inCurveA : first input curve. IEngineCurve inCurveB : second input curve. IEngineCurve outCurve : output curve.
Return	double : new calculated actual Y-value.
Description	Set the output level in the actual output curve[0] Y-value as a logic OR combination from actual input curve[0] A Y-value and actual input curve[0] B Y-value. incurveA inCurveB outCurve 0 0 0 1 0 1 0 1 1 1 1 1 Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves.
Example	var yVal = EngineLibrary.AnalogLogic.ActOR(aKey, IN_MY_CURVE_A, IN_MY_CURVE_B, OUT_MY_CURVE);

Function	double ActOR(int key, IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)
Parameter	int key : object key from CreateAnalogLogic(). IEngineCurve inCurveA : first input curve. int inCurveIndexA : first input curve index. IEngineCurve inCurveB : second input curve. int inCurveIndexB : second input curve index. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.
Return	double : new calculated actual Y-value.
Description	Set the output level in the actual output curve[outCurveIndex] Y-value as a logic OR combination from actual input curve[intCurveIndexA] A Y-value and actual input curve[inCurveIndexB] B Y-value. incurveA inCurveB outCurve 0 0 0 1 0 1 0 1 1 1 1 1 Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves.
Example	var yVal = EngineLibrary.AnalogLogic.ActOR(aKey, IN_MY_CURVE_A, 2, IN_MY_CURVE_B, 3, OUT_MY_CURVE, 0);



5.15.1.14 ActNOR()

Function	double ActNOR(int key, IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)
Parameter	int key : object key from CreateAnalogLogic(). IEngineCurve inCurveA : first input curve. IEngineCurve inCurveB : second input curve. IEngineCurve outCurve : output curve.
Return	double : new calculated actual Y-value.
Description	Set the output level in the actual output curve[0] Y-value as a logic NOR combination from actual input curve[0] A Y-value and actual input curve[0] B Y-value. incurveA inCurveB outCurve 0 0 1 1 0 0 0 1 0 1 1 0 Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves.
Example	var yVal = EngineLibrary.AnalogLogic.ActNOR(aKey, IN_MY_CURVE_A, IN_MY_CURVE_B, OUT_MY_CURVE);

Function	double ActNOR(int key, IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)
Parameter	int key : object key from CreateAnalogLogic(). IEngineCurve inCurveA : first input curve. int inCurveIndexA : first input curve index. IEngineCurve inCurveB : second input curve. int inCurveIndexB : second input curve index. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.
Return	double : new calculated actual Y-value.
Description	Set the output level in the actual output curve[outCurveIndex] Y-value as a logic NOR combination from actual input curve[intCurveIndexA] A Y-value and actual input curve[inCurveIndexB] B Y-value. incurveA inCurveB outCurve 0 0 1 1 0 0 0 1 0 1 1 0 Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves.
Example	var yVal = EngineLibrary.AnalogLogic.ActNOR(aKey, IN_MY_CURVE_A, 2, IN_MY_CURVE_B, 3, OUT_MY_CURVE, 0);



5.15.1.15 ActXOR()

Function	double ActXOR(int key, IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)
Parameter	int key : object key from CreateAnalogLogic(). IEngineCurve inCurveA : first input curve. IEngineCurve inCurveB : second input curve. IEngineCurve outCurve : output curve.
Return	double : new calculated actual Y-value.
Description	Set the output level in the actual output curve[0] Y-value as a logic XOR combination from actual input curve[0] A Y-value and actual input curve[0] B Y-value. incurveA inCurveB outCurve 0 0 0 1 0 1 0 1 1 1 1 0 Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves.
Example	var yVal = EngineLibrary.AnalogLogic.ActXOR(aKey, IN_MY_CURVE_A, IN_MY_CURVE_B, OUT_MY_CURVE);

Function	double ActXOR(int key, IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)
Parameter	int key : object key from CreateAnalogLogic(). IEngineCurve inCurveA : first input curve. int inCurveIndexA : first input curve index. IEngineCurve inCurveB : second input curve. int inCurveIndexB : second input curve index. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.
Return	double : new calculated actual Y-value.
Description	Set the output level in the actual output curve[outCurveIndex] Y-value as a logic XOR combination from actual input curve[intCurveIndexA] A Y-value and actual input curve[inCurveIndexB] B Y-value. incurveA inCurveB outCurve 0 0 0 1 0 1 0 1 1 1 1 0 Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves.
Example	var yVal = EngineLibrary.AnalogLogic.ActXOR(aKey, IN_MY_CURVE_A, 2, IN_MY_CURVE_B, 3, OUT_MY_CURVE, 0);

**5.15.1.16 ActAND()**

Function	double ActAND(int key, IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)
Parameter	int key : object key from CreateAnalogLogic(). IEngineCurve inCurveA : first input curve. IEngineCurve inCurveB : second input curve. IEngineCurve outCurve : output curve.
Return	double : new calculated actual Y-value.
Description	Set the output level in the actual output curve[0] Y-value as a logic AND combination from actual input curve[0] A Y-value and actual input curve[0] B Y-value. incurveA inCurveB outCurve 0 0 0 1 0 0 0 1 0 1 1 1 Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves.
Example	var yVal = EngineLibrary.AnalogLogic.ActAND(aKey, IN_MY_CURVE_A, IN_MY_CURVE_B, OUT_MY_CURVE);

Function	<pre>double ActAND(int key, IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)</pre>															
Parameter	<pre>int key : object key from CreateAnalogLogic(). IEngineCurve inCurveA : first input curve. int inCurveIndexA : first input curve index. IEngineCurve inCurveB : second input curve. int inCurveIndexB : second input curve index. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.</pre>															
Return	<pre>double : new calculated actual Y-value.</pre>															
Description	Set the output level in the actual output curve[outCurveIndex] Y-value as a logic AND combination from actual input curve[intCurveIndexA] A Y-value and actual input curve[inCurveIndexB] B Y-value. <table><tr><td>incurveA</td><td>inCurveB</td><td>outCurve</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table> Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves.	incurveA	inCurveB	outCurve	0	0	0	1	0	0	0	1	0	1	1	1
incurveA	inCurveB	outCurve														
0	0	0														
1	0	0														
0	1	0														
1	1	1														
Example	<pre>var yVal = EngineLibrary.AnalogLogic.ActAND(aKey, IN_MY_CURVE_A, 2, IN_MY_CURVE_B, 3, OUT_MY_CURVE, 0);</pre>															



5.15.1.17 ActNAND()

Function	double ActNAND(int key, IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)
Parameter	int key : object key from CreateAnalogLogic(). IEngineCurve inCurveA : first input curve. IEngineCurve inCurveB : second input curve. IEngineCurve outCurve : output curve.
Return	double : new calculated actual Y-value.
Description	Set the output level in the actual output curve[0] Y-value as a logic NAND combination from actual input curve[0] A Y-value and actual input curve[0] B Y-value. incurveA inCurveB outCurve 0 0 1 1 0 1 0 1 1 1 1 0 Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves.
Example	var yVal = EngineLibrary.AnalogLogic.ActNAND(aKey, IN_MY_CURVE_A, IN_MY_CURVE_B, OUT_MY_CURVE);

Function	double ActNAND(int key, IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)
Parameter	int key : object key from CreateAnalogLogic(). IEngineCurve inCurveA : first input curve. int inCurveIndexA : first input curve index. IEngineCurve inCurveB : second input curve. int inCurveIndexB : second input curve index. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.
Return	double : new calculated actual Y-value.
Description	Set the output level in the actual output curve[outCurveIndex] Y-value as a logic NAND combination from actual input curve[intCurveIndexA] A Y-value and actual input curve[inCurveIndexB] B Y-value. incurveA inCurveB outCurve 0 0 1 1 0 1 0 1 1 1 1 0 Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves.
Example	var yVal = EngineLibrary.AnalogLogic.ActNAND(aKey, IN_MY_CURVE_A, 2, IN_MY_CURVE_B, 3, OUT_MY_CURVE, 0);

**5.15.1.18 ActStream()**

Function	<pre>double ActStream(int key, bool logic, bool msbFirst, IEngineCurve outCurve)</pre>
Parameter	<pre>int key : object key from CreateAnalogLogic(). bool logic : True - 1 is high, 0 is low. False - 1 is low, 0 is high. bool msbFirst : True - MSB byte first. False - LSB byte first. IEngineCurve outCurve : output curve.</pre>
Return	<pre>double : new calculated actual Y-value.</pre>
Description	<pre>Set a created stream object to the actual curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves.</pre>
Example	<pre>var yVal = EngineLibrary.AnalogLogic.ActStream(aKey, true, false, OUT_MY_CURVE);</pre>

Function	<pre>double ActStream(int key, bool logic, bool msbFirst, IEngineCurve outCurve)</pre>
Parameter	<pre>int key : object key from CreateAnalogLogic(). bool logic : True - 1 is high, 0 is low. False - 1 is low, 0 is high. bool msbFirst : True - MSB byte first. False - LSB byte first. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.</pre>
Return	<pre>double : new calculated actual Y-value.</pre>
Description	<pre>Set a created stream object to the actual curve[outCurveIndex] Y-value. Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves.</pre>
Example	<pre>var yVal = EngineLibrary.AnalogLogic.ActStream(aKey, true, false, OUT_MY_CURVE, 5);</pre>



5.15.2 Filter

5.15.2.1 CreateAnalogPolynom()

Function	int CreateAnalogPolynom(double[] A, double[] B)
Parameter	double[] A : numerator coefficients double[] B : denominator coefficients
Return	int : object key for AnalogPolynom.
Description	Create an AnalogPolynom object for input and output curves. Gain is 1.0 and nominal frequency is 1.0Hz.
Example	var aKey = EngineLibrary.Filter.CreateAnalogPolynom(new double[]{1.0}, new double[]{1.0, 0.5, 2.0});

Function	int CreateAnalogPolynom(double[] A, double[] B, double gain, double frequency)
Parameter	double[] A : numerator coefficients double[] B : denominator coefficients double gain : gain factor double frequency : norm frequency.
Return	int : object key for AnalogPolynom.
Description	Create an AnalogPolynom object for input and output curves.
Example	var aKey = EngineLibrary.Filter.CreateAnalogPolynom(new double[]{1.0}, new double[]{1.0, 0.5, 2.0}, 2.0, 0.1);

$$A(P) = gain * \frac{A0 + A1P + A2P^2 + A3P^3 + A4P^4 \dots}{B0 + B1P + B2P^2 + B3P^3 + B4P^4 \dots}$$



5.15.2.2 CreateFIR()

Function	int CreateFIR(double[] az, double gain)
Parameter	double[] az : coefficients buffer. double gain : filter gain.
Return	int : object key for FIR filter.
Description	Create a digital Finite Impulse Response (FIR) filter polynom with a gain. Sample time is equal to system tick time.
Example	var aKey = EngineLibrary.Filter.CreateFIR(new double[]{1.5, 1.5, 1.5, 1.5, 1.5}, 5.0);

Function	int CreateFIR(double[] az, double gain, double sampleTime)
Parameter	double[] az : coefficients buffer. double gain : filter gain. double sampleTime : filter sample time.
Return	int : object key for FIR filter.
Description	Create a digital Finite Impulse Response (FIR) filter polynom with a gain and a defined sample time.
Example	var aKey = EngineLibrary.Filter.CreateFIR(new double[]{1.5, 1.5, 1.5, 1.5, 1.5}, 5.0, 1E-6);

$$H(z) = gain * \sum_{k=0}^{N-1} a[k] * z^{-k}$$



5.15.2.3 CreateIIR()

Function	<code>int CreateIIR(double[] az, double[] bz, double gain)</code>
Parameter	<code>double[] az</code> : a coefficients buffer. <code>double[] bz</code> : b coefficients buffer. <code>double gain</code> : filter gain.
Return	<code>int</code> : object key for FIR filter.
Description	Create a digital Infinit Impulse Response (IIR) filter polynom with a gain. Sample time is equal to system tick time. <code>bz[0]</code> must be 1.
Example	<pre>var aKey = EngineLibrary.Filter.CreateFIR(new double[]{1.5, 1.5, 1.5, 1.5, 1.5}, new double[]{0.5, 0.5, 0.5, 0.5, 0.5}, 5.0);</pre>

Function	<code>int CreateIIR(double[] az, double[] bz, double gain, double sampleTime)</code>
Parameter	<code>double[] az</code> : a coefficients buffer. <code>double[] bz</code> : b coefficients buffer. <code>double gain</code> : filter gain. <code>double sampleTime</code> : filter sampleTime
Return	<code>int</code> : object key for FIR filter.
Description	Create a digital Infinit Impulse Response (IIR) filter polynom with a gain and a defined sample time. <code>bz[0]</code> must be 1.
Example	<pre>var aKey = EngineLibrary.Filter.CreateFIR(new double[]{1.5, 1.5, 1.5, 1.5, 1.5}, new double[]{0.5, 0.5, 0.5, 0.5, 0.5}, 5.0, 1E-6);</pre>

$$H(z) = gain * \frac{\sum_{k=0}^M a[k] * z^{-k}}{\sum_{k=1}^N b[k] * z^{-k}}$$



5.15.2.4 CurveFilterAmplitudeResponse()

Function	<code>void CurveFilterAmplitudeResponse(int key, IEngineCurve outCurve)</code>
Parameter	<code>int key</code> : filter object key (Analog, FIR, IIR). <code>IEngineCurve outcurve</code> : output curve.
Return	-
Description	Calculate the filter amplitude response to an output curve[0]. For user curves only. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Filter.CurveFilterAmplitudeResponse(aKey OUT_MY_CURVE);</code>

Function	<code>void CurveFilterAmplitudeResponse(int key, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>int key</code> : filter object key (Analog, FIR, IIR). <code>IEngineCurve outcurve</code> : output curve. <code>int outCurveIndex</code> : output curve index.
Return	-
Description	Calculate the filter amplitude response to an output curve[outCurveIndex]. For user curves only. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Filter.CurveFilterAmplitudeResponse(aKey OUT_MY_CURVE, 3);</code>

$$g(f) = \text{Filter}(f)$$



5.15.2.5 CurveFilterPhaseResponse()

Function	<code>void CurveFilterPhaseResponse(int key, IEngineCurve outCurve)</code>
Parameter	<code>int key</code> : filter object key (Analog, FIR, IIR). <code>IEngineCurve outcurve</code> : output curve.
Return	-
Description	Calculate the filter phase response to an output curve[0]. For user curves only. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Filter.CurveFilterPhaseResponse(aKey OUT_MY_CURVE);</code>

Function	<code>void CurveFilterPhaseResponse(int key, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>int key</code> : filter object key (Analog, FIR, IIR). <code>IEngineCurve outcurve</code> : output curve. <code>int outCurveIndex</code> : output curve index.
Return	-
Description	Calculate the filter phase response to an output curve[outCurveIndex]. For user curves only. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Filter.CurveFilterPhaseResponse(aKey OUT_MY_CURVE, 3);</code>

$$g(\varphi) = \text{Filter}(\varphi)$$



5.15.2.6 ActFiltering()

Function	<pre>double ActFiltering(int key, IEngineCurve inCurve, IEngineCurve outCurve)</pre>
Parameter	<pre>int key : filter object key. IEngineCurve inCurve : input curve. IEngineCurve outCurve : output curve.</pre>
Return	<pre>double : new calculated actual Y-value.</pre>
Description	Calculate the actual output curve[0] Y-value from an actual input curve[0] Y-value with a created filter object. Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var yVal = EngineLibrary.Filter.ActFiltering(key, IN_MY_CURVE, OUT_MY_CURVE);</pre>

Function	<pre>double ActFiltering(int key, IEngineCurve inCurve, int inCurveIndex, IEngineCurve outCurve, int outCurveIndex)</pre>
Parameter	<pre>int key : filter object key. IEngineCurve inCurve : input curve. int inCurveIndex : input curve index. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.</pre>
Return	<pre>double : new calculated actual Y-value.</pre>
Description	Calculate the actual output curve[outCurveIndex] Y-value from an actual input curve[inCurveIndex] Y-value with a created filter object. Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var yVal = EngineLibrary.Filter.ActFiltering(key, IN_MY_CURVE, 1 OUT_MY_CURVE, 1);</pre>

$$g(x[a]) = f(x[a] * Filter$$



5.15.3 Operation

5.15.3.1 CreateIntegration()

Function	<code>int CreateIntegration(double c)</code>
Parameter	<code>double c</code> : integration start value.
Return	<code>int</code> : object key for integration.
Description	Create an integration object with a start value.
Example	<code>var aKey = EngineLibrary.Operation.CreateIntegration(2.0);</code>

5.15.3.2 CreateRMA()

Function	<code>int CreateRMA(int windowSize)</code>
Parameter	<code>int windowSize</code> : RMA windowSize.
Return	<code>int</code> : object key for RMA.
Description	Create a running moving average (RMA) object for double with a window size.
Example	<code>var aKey = EngineLibrary.Operation.CreateRMA(100);</code>

5.15.3.3 CreateRMAInt()

Function	<code>int CreateRMAInt(int windowSize)</code>
Parameter	<code>int windowSize</code> : RMA windowSize.
Return	<code>int</code> : object key for RMA.
Description	Create a running moving average (RMA) object for Int64 with a window size.
Example	<code>var aKey = EngineLibrary.Operation.CreateRMAInt(100);</code>

5.15.3.4 CreateRMS()

Function	<code>int CreateRMS(int windowSize)</code>
Parameter	<code>int windowSize</code> : RMS windowSize.
Return	<code>int</code> : object key for RMS.
Description	Create a root mean square (RMS) object for double with a window size.
Example	<code>var aKey = EngineLibrary.Operation.CreateRMS(100);</code>



5.15.3.5 CurveDerivative()

Function	<code>void CurveDerivative(IEngineCurve inCurve, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurve</code> : input curve. <code>IEngineCurve outCurve</code> : output curve.
Return	-
Description	Create the first derivative of an input curve[0] to an output curve[0]. The entire output curve[0] is calculated at once. Use in <code>Init()</code> or <code>Post()</code> functions.
Example	<code>EngineLibrary.Operation.CurveDerivative(IN_MY_CURVE, OUT_MY_CURVE);</code>

Function	<code>void CurveDerivative(IEngineCurve inCurve, int inCurveIndex, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>IEngineCurve inCurve</code> : input curve. <code>int inCurveIndex</code> : input curve index. <code>IEngineCurve outCurve</code> : output curve. <code>int outCurveIndex</code> : output curve index.
Return	-
Description	Create the first derivative of an input curve[inCurveIndex] to an output curve[outCurveIndex]. The entire output curve[outCurveIndex] is calculated at once. Use in <code>Init()</code> or <code>Post()</code> functions.
Example	<code>EngineLibrary.Operation.CurveDerivative(IN_MY_CURVE, 2, OUT_MY_CURVE, 0);</code>

$$f(x[a]) = \frac{g(x[a]) - g(x[a - 1])}{x[a] - x[a - 1]}$$

**5.15.3.6 CurveIntegration()**

Function	<code>void CurveIntegratation(int key, IEngineCurve inCurve, IEngineCurve outCurve)</code>
Parameter	<code>int key</code> : integration object key. <code>IEngineCurve inCurve</code> : input curve. <code>IEngineCurve outCurve</code> : output curve.
Return	-
Description	Create the integration of an input curve[0] to an output curve[0] with the parameter of the created integration object. The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveIntegration(aKey, IN_MY_CURVE, OUT_MY_CURVE);</code>

Function	<code>void CurveIntegration(int key, IEngineCurve inCurve, int inCurveIndex, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>int key</code> : integration object key. <code>IEngineCurve inCurve</code> : input curve. <code>int inCurveIndex</code> : input curve index. <code>IEngineCurve outCurve</code> : output curve. <code>int outCurveIndex</code> : output curve index.
Return	-
Description	Create the integration of an input curve[inCurveIndex] to an output curve[outCurveIndex] with the parameter of the created integration object. The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveIntegration(aKey, IN_MY_CURVE, 2, OUT_MY_CURVE, 1);</code>

$$f(x[a]) = C + \sum_{i=0}^a \frac{g(x[i]) + g(x[i-1])}{2} * (x[i] - x[i-1])$$



5.15.3.7 CurveRMA()

Function	<code>double CurveRMA(int key, IEngineCurve inCurve, IEngineCurve outCurve)</code>
Parameter	<code>int key</code> : RMA object key. <code>IEngineCurve inCurve</code> : input curve. <code>IEngineCurve outCurve</code> : output curve (can be null).
Return	<code>double</code> : average value of the entire input curve[0].
Description	Create a running moving average (RMA) of an input curve[0] to an output curve[0] with the parameters from the created RMA object. The entire output curve[0] is calculated at once. Use in <code>Init()</code> or <code>Post()</code> functions.
Example	<pre>var dAve = EngineLibrary.Operation.CurveRMA(aKey, IN_MY_CURVE, OUT_MY_CURVE);</pre>

Function	<code>double CurveRMA(int key, IEngineCurve inCurve, int inCurveIndex, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>int key</code> : RMA object key. <code>IEngineCurve inCurve</code> : input curve. <code>int inCurveIndex</code> : input curve index. <code>IEngineCurve outCurve</code> : output curve (can be null). <code>int outCurveIndex</code> : output curve index.
Return	<code>double</code> : average value of the entire input curve[inCurveIndex].
Description	Create the running moving average (RMA) of an input curve[inCurveIndex] to an output curve[outCurveIndex] with the parameters of the created RMA object. The entire output curve[outCurveIndex] is calculated at once. Use in <code>Init()</code> or <code>Post()</code> functions.
Example	<pre>var dAve = EngineLibrary.Operation.CurveRMA(aKey, IN_MY_CURVE, 4, OUT_MY_CURVE, 5);</pre>

$$f(x[a]) = \frac{1}{N} \sum_{i=a}^{a-N} g(x[i])$$



5.15.3.8 CurveRMAInt()

Function	<code>Int64 CurveRMAInt(int key, IEngineCurve inCurve, IEngineCurve outCurve)</code>
Parameter	<code>int key</code> : RMA object key. <code>IEngineCurve inCurve</code> : input curve. <code>IEngineCurve outCurve</code> : output curve (can be null).
Return	<code>Int64</code> : average integer value of the entire input curve[0].
Description	Create a running moving average (RMA) of an input curve[0] to an output curve[0] with the parameters from the created RMA object. The entire output curve[0] is calculated at once. Use in <code>Init()</code> or <code>Post()</code> functions.
Example	<pre>var iAve = EngineLibrary.Operation.CurveRMAInt(aKey, IN_MY_CURVE, OUT_MY_CURVE);</pre>

Function	<code>Int64 CurveRMAInt(int key, IEngineCurve inCurve, int inCurveIndex, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>int key</code> : RMA object key. <code>IEngineCurve inCurve</code> : input curve. <code>int inCurveIndex</code> : input curve index. <code>IEngineCurve outCurve</code> : output curve (can be null). <code>int outCurveIndex</code> : output curve index.
Return	<code>Int64</code> : average integer value of the entire input curve[inCurveIndex].
Description	Create the running moving average (RMA) of an input curve[inCurveIndex] to an output curve[outCurveIndex] with the parameters of the created RMA object. The entire output curve[outCurveIndex] is calculated at once. Use in <code>Init()</code> or <code>Post()</code> functions.
Example	<pre>var iAve = EngineLibrary.Operation.CurveRMAInt(aKey, IN_MY_CURVE, 4, OUT_MY_CURVE, 5);</pre>

**5.15.3.9 CurveRMS()**

Function	<code>double CurveRMS(int key, IEngineCurve inCurve, IEngineCurve outCurve)</code>
Parameter	<code>int key</code> : RMS object key. <code>IEngineCurve inCurve</code> : input curve. <code>IEngineCurve outCurve</code> : output curve (can be null).
Return	<code>double</code> : rms value of the entire input curve[0].
Description	Create a root mean square (RMS) of an input curve[0] to an output curve[0] with the parameters from the created RMS object. The entire output curve[0] is calculated at once. Use in <code>Init()</code> or <code>Post()</code> functions.
Example	<pre>var dRms = EngineLibrary.Operation.CurveRMS(aKey, IN_MY_CURVE, OUT_MY_CURVE);</pre>

Function	<code>double CurveRMS(int key, IEngineCurve inCurve, int inCurveIndex, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>int key</code> : RMS object key. <code>IEngineCurve inCurve</code> : input curve. <code>int inCurveIndex</code> : input curve index. <code>IEngineCurve outCurve</code> : output curve (can be null). <code>int outCurveIndex</code> : output curve index.
Return	<code>double</code> : RMS value of the entire input curve[inCurveIndex].
Description	Create the root mean square (RMS) of an input curve[inCurveIndex] to an output curve[outCurveIndex] with the parameters of the created RMS object. The entire output curve[outCurveIndex] is calculated at once. Use in <code>Init()</code> or <code>Post()</code> functions.
Example	<pre>var dRms = EngineLibrary.Operation.CurveRMS(aKey, IN_MY_CURVE, 4, OUT_MY_CURVE, 5);</pre>

$$f(x[a]) = \sqrt{\frac{1}{N} \sum_{i=a}^{a-N} g(x[i])^2}$$



5.15.3.10 CurveAdd()

Function	<code>void CurveAdd(IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. IEngineCurve inCurveB : input curve B. IEngineCurve outCurve : output curve.</code>
Return	-
Description	Addition of input curve[0] A and input curve[0] B to output curve[0]. The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveAdd(IN_MY_CURVE_A, IN_MY_CURVE_B OUT_MY_CURVE);</code>

Function	<code>void CurveAdd(IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. int inCurveIndexA : input curve index A. IEngineCurve inCurveB : input curve B. int inCurveIndexB : input curve index B. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.</code>
Return	-
Description	Addition of input curve[inCurveIndexA] A and input curve[inCurveIndexB] B to output curve[outCurveIndex]. The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveAdd(IN_MY_CURVE_A, 2, IN_MY_CURVE_B 3, OUT_MY_CURVE 5);</code>

$$f(x[a]) = g_A(x[a]) + g_B(x[a])$$



5.15.3.11 CurveAddInt()

Function	<code>void CurveAddInt(IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. IEngineCurve inCurveB : input curve B. IEngineCurve outCurve : output curve.</code>
Return	-
Description	Addition of integer input curve[0] A and integer input curve[0] B to integer output curve[0]. The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveAddInt(IN_MY_CURVE_A, IN_MY_CURVE_B OUT_MY_CURVE);</code>

Function	<code>void CurveAddInt(IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. int inCurveIndexA : input curve index A. IEngineCurve inCurveB : input curve B. int inCurveIndexB : input curve index B. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.</code>
Return	-
Description	Addition of integer input curve[inCurveIndexA] A and integer input curve[inCurveIndexB] B to integer output curve[outCurveIndex]. The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveAddInt(IN_MY_CURVE_A, 2, IN_MY_CURVE_B 3, OUT_MY_CURVE 5);</code>

**5.15.3.12 CurveSum()**

Function	<code>void CurveSum(IEngineCurve inCurve, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurve</code> : input curve. <code>IEngineCurve outCurve</code> : output curve.
Return	-
Description	Summation of all input curve[0...n] to output curve[0]. The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveSum(IN_MY_CURVE, OUT_MY_CURVE);</code>

Function	<code>void CurveSum(IEngineCurve inCurve, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>IEngineCurve inCurve</code> : input curve. <code>IEngineCurve outCurve</code> : output curve. <code>int outCurveIndex</code> : output curve index.
Return	-
Description	Summation of all input curve[0...n] to output curve[outCurveIndex]. The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveSum(IN_MY_CURVE, OUT_MY_CURVE, 1);</code>

$$f(x[a]) = \sum_{i=1}^n g_i(x[a])$$



5.15.3.13 CurveSumInt()

Function	<code>void CurveSumInt(IEngineCurve inCurve, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurve</code> : input curve. <code>IEngineCurve outCurve</code> : output curve.
Return	-
Description	Summation of all integer input curve[0...n] to integer output curve[0]. The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveSumInt(IN_MY_CURVE, OUT_MY_CURVE);</code>

Function	<code>void CurveSumInt(IEngineCurve inCurve, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>IEngineCurve inCurve</code> : input curve. <code>IEngineCurve outCurve</code> : output curve. <code>int outCurveIndex</code> : output curve index.
Return	-
Description	Summation of all integer input curve[0...n] to integer output curve[outCurveIndex]. The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveSumInt(IN_MY_CURVE, OUT_MY_CURVE, 1);</code>

**5.15.3.14 CurveDiff()**

Function	<code>void CurveDiff(IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. IEngineCurve inCurveB : input curve B. IEngineCurve outCurve : output curve.</code>
Return	-
Description	Difference between input curve[0] A and input curve[0] B to output curve[0]. The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveDiff(IN_MY_CURVE_A, IN_MY_CURVE_B OUT_MY_CURVE);</code>

Function	<code>void CurveDiff(IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. int inCurveIndexA : input curve index A. IEngineCurve inCurveB : input curve B. int inCurveIndexB : input curve index B. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.</code>
Return	-
Description	Difference between input curve[inCurveIndexA] A and input curve[inCurveIndexB] B to output curve[outCurveIndex]. The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveDiff(IN_MY_CURVE_A, 2, IN_MY_CURVE_B 3, OUT_MY_CURVE 5);</code>

$$f(x[a]) = g_A(x[a]) - g_B(x[a])$$

**5.15.3.15 CurveDiffInt()**

Function	<code>void CurveDiffInt(IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. IEngineCurve inCurveB : input curve B. IEngineCurve outCurve : output curve.</code>
Return	-
Description	Difference between integer input curve[0] A and integer input curve[0] B to integer output curve[0]. The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveDiffInt(IN_MY_CURVE_A, IN_MY_CURVE_B OUT_MY_CURVE);</code>

Function	<code>void CurveDiffInt(IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. int inCurveIndexA : input curve index A. IEngineCurve inCurveB : input curve B. int inCurveIndexB : input curve index B. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.</code>
Return	-
Description	Difference between integer input curve[inCurveIndexA] A and integer input curve[inCurveIndexB] B to integer output curve[outCurveIndex]. The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveDiffInt(IN_MY_CURVE_A, 2, IN_MY_CURVE_B 3, OUT_MY_CURVE 5);</code>



5.15.3.16 CurveMul()

Function	<code>void CurveMul(IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. IEngineCurve inCurveB : input curve B. IEngineCurve outCurve : output curve.</code>
Return	-
Description	Multiplication of input curve[0] A and input curve[0] B to output curve[0]. The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	<pre>EngineLibrary.Operation.CurveMul(IN_MY_CURVE_A, IN_MY_CURVE_B OUT_MY_CURVE);</pre>

Function	<code>void CurveMul(IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. int inCurveIndexA : input curve index A. IEngineCurve inCurveB : input curve B. int inCurveIndexB : input curve index B. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.</code>
Return	-
Description	Multiplication of input curve[inCurveIndexA] A and input curve[inCurveIndexB] B to output curve[outCurveIndex]. The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	<pre>EngineLibrary.Operation.CurveMul(IN_MY_CURVE_A, 2, IN_MY_CURVE_B 3, OUT_MY_CURVE 5);</pre>

$$f(x[a]) = g_A(x[a]) * g_B(x[a])$$

**5.15.3.17 CurveMulInt()**

Function	<code>void CurveMulInt(IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. IEngineCurve inCurveB : input curve B. IEngineCurve outCurve : output curve.</code>
Return	-
Description	Multiplication of integer input curve[0] A and integer input curve[0] B to integer output curve[0]. The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveMulInt(IN_MY_CURVE_A, IN_MY_CURVE_B OUT_MY_CURVE);</code>

Function	<code>void CurveMulInt(IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. int inCurveIndexA : input curve index A. IEngineCurve inCurveB : input curve B. int inCurveIndexB : input curve index B. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.</code>
Return	-
Description	Multiplication of integer input curve[inCurveIndexA] A and integer input curve[inCurveIndexB] B to integer output curve[outCurveIndex]. The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveMulInt(IN_MY_CURVE_A, 2, IN_MY_CURVE_B 3, OUT_MY_CURVE 5);</code>

**5.15.3.18 CurveProduct()**

Function	<code>void CurveProduct(IEngineCurve inCurve, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurve</code> : input curve. <code>IEngineCurve outCurve</code> : output curve.
Return	-
Description	Product of all input curve[0...n] to output curve[0]. The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveProduct(IN_MY_CURVE, OUT_MY_CURVE);</code>

Function	<code>void CurveProduct(IEngineCurve inCurve, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>IEngineCurve inCurve</code> : input curve. <code>IEngineCurve outCurve</code> : output curve. <code>int outCurveIndex</code> : output curve index.
Return	-
Description	Product of all input curve[0...n] to output curve[outCurveIndex]. The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveProduct(IN_MY_CURVE, OUT_MY_CURVE, 1);</code>

$$f(x[a]) = \prod_{i=1}^n g_i(x[a])$$



5.15.3.19 CurveProductInt()

Function	<code>void CurveProductInt(IEngineCurve inCurve, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurve</code> : input curve. <code>IEngineCurve outCurve</code> : output curve.
Return	-
Description	Product of all integer input curve[0...n] to integer output curve[0]. The entire output curve[0] is calculated at once. Use in <code>Init()</code> or <code>Post()</code> functions.
Example	<code>EngineLibrary.Operation.CurveProductInt(IN_MY_CURVE, OUT_MY_CURVE);</code>

Function	<code>void CurveProductInt(IEngineCurve inCurve, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>IEngineCurve inCurve</code> : input curve. <code>IEngineCurve outCurve</code> : output curve. <code>int outCurveIndex</code> : output curve index.
Return	-
Description	Product of all integer input curve[0...n] to integer output curve[outCurveIndex]. The entire output curve[outCurveIndex] is calculated at once. Use in <code>Init()</code> or <code>Post()</code> functions.
Example	<code>EngineLibrary.Operation.CurveProductInt(IN_MY_CURVE, OUT_MY_CURVE, 1);</code>



5.15.3.20 CurveDiv()

Function	<code>void CurveDiv(IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. IEngineCurve inCurveB : input curve B. IEngineCurve outCurve : output curve.</code>
Return	-
Description	Division between input curve[0] A and input curve[0] B to output curve[0]. The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveDiv(IN_MY_CURVE_A, IN_MY_CURVE_B OUT_MY_CURVE);</code>

Function	<code>void CurveDiv(IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. int inCurveIndexA : input curve index A. IEngineCurve inCurveB : input curve B. int inCurveIndexB : input curve index B. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.</code>
Return	-
Description	Division between input curve[inCurveIndexA] A and input curve[inCurveIndexB] B to output curve[outCurveIndex]. The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveDiv(IN_MY_CURVE_A, 2, IN_MY_CURVE_B 3, OUT_MY_CURVE 5);</code>

$$f(x[a]) = \frac{g_A(x[a])}{g_B(x[a])}$$

**5.15.3.21 CurveDivInt()**

Function	<code>void CurveDivInt(IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. IEngineCurve inCurveB : input curve B. IEngineCurve outCurve : output curve.</code>
Return	-
Description	Division between integer input curve[0] A and integer input curve[0] B to integer output curve[0]. The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveDivInt(IN_MY_CURVE_A, IN_MY_CURVE_B OUT_MY_CURVE);</code>

Function	<code>void CurveDivInt(IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. int inCurveIndexA : input curve index A. IEngineCurve inCurveB : input curve B. int inCurveIndexB : input curve index B. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.</code>
Return	-
Description	Division between integer input curve[inCurveIndexA] A and integer input curve[inCurveIndexB] B to integer output curve[outCurveIndex]. The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveDivInt(IN_MY_CURVE_A, 2, IN_MY_CURVE_B 3, OUT_MY_CURVE 5);</code>



5.15.3.22 CurveFactorOffset()

Function	<code>void CurveFactorOffset(IEngineCurve inCurve, IEngineCurve outCurve, double factor, double offset)</code>
Parameter	<code>IEngineCurve incurve : input curve. IEngineCurve outCurve : output curve. double factor : factor value. double offset : offset value.</code>
Return	-
Description	Multiply a Factor and add an Offset from input curve[0] to output curve[0]. The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveFactorOffset(IN_MY_CURVE, OUT_MY_CURVE, 2.0, -10.0);</code>

Function	<code>void CurveFactorOffset(IEngineCurve inCurve, int inCurveIndex, IEngineCurve outCurve, int outCurveIndex, double factor, double offset)</code>
Parameter	<code>IEngineCurve inCurve : input curve. int inCurveIndex : input curve index. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index. double factor : factor value. double offset : offset value.</code>
Return	-
Description	Multiply a Factor and add an Offset from input curve[inCurveIndex] to output curve[outCurveIndex]. The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveFactorOffset(IN_MY_CURVE, 6, OUT_MY_CURVE, 1, 2.0, -10.0);</code>

$$f(x[a]) = g_A(x[a]) * factor + offset$$



5.15.3.23 CurveFactorOffsetInt()

Function	<code>void CurveFactorOffsetInt(IEngineCurve inCurve, IEngineCurve outCurve, Int64 factor, Int64 offset)</code>
Parameter	<code>IEngineCurve incurve : input curve. IEngineCurve outCurve : output curve. Int64 factor : factor value. Int64 offset : offset value.</code>
Return	-
Description	Multiply an integer Factor and add an integer Offset from integer input curve[0] to integer output curve[0]. The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	<pre>EngineLibrary.Operation.CurveFactorOffsetInt(IN_MY_CURVE, OUT_MY_CURVE, 2, -10);</pre>

Function	<code>void CurveFactorOffsetInt(IEngineCurve inCurve, int inCurveIndex, IEngineCurve outCurve, int outCurveIndex, Int64 factor, Int64 offset)</code>
Parameter	<code>IEngineCurve inCurve : input curve. int inCurveIndex : input curve index. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index. Int64 factor : factor value. Int64 offset : offset value.</code>
Return	-
Description	Multiply an integer Factor and add an integer Offset from integer input curve[inCurveIndex] to integer output curve[outCurveIndex]. The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	<pre>EngineLibrary.Operation.CurveFactorOffsetInt(IN_MY_CURVE, 6, OUT_MY_CURVE, 1, 2, -10);</pre>

**5.15.3.24 CurveConvertDB()**

Function	<code>void CurveConvertDB(IEngineCurve inCurve, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurve</code> : input curve. <code>IEngineCurve outCurve</code> : output curve.
Return	-
Description	Convert Y-values of <code>inCurve[0]</code> to dezibel (DB) y-values of <code>outcurve[0]</code> . The entire output curve[0] is calculated at once. Use in <code>Init()</code> or <code>Post()</code> functions.
Example	<code>EngineLibrary.Operation.CurveConvertDB(IN_MY_CURVE, OUT_MY_CURVE);</code>

Function	<code>void CurveConvertDB(IEngineCurve inCurve, int inCurveIndex, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>IEngineCurve inCurve</code> : input curve. <code>int inCurveIndex</code> : input curve index. <code>IEngineCurve outCurve</code> : output curve. <code>int outCurveIndex</code> : output curve index.
Return	-
Description	Convert Y-values of <code>inCurve[inCurveIndex]</code> to dezibel (DB) y-values of <code>outcurve[outCurveIndex]</code> . The entire output curve[outCurveIndex] is calculated at once. Use in <code>Init()</code> or <code>Post()</code> functions.
Example	<code>EngineLibrary.Operation.CurveConvertDB(IN_MY_CURVE, 3 OUT_MY_CURVE, 2);</code>

$$f(x[a]) = 20 * \log_{10}(g_A(x[a]))$$

**5.15.3.25 CurveComposition()**

Function	<code>void CurveComposition(IEngineCurve inCurve, IEngineCurve compCurve, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurve : input curve. IEngineCurve compCurve : composition curve. IEngineCurve outCurve : output curve.</code>
Return	-
Description	Composite output curve[0] from an input curve[0] and a composite curve[0]. The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveComposition(IN_MY_CURVE, IN_MY_CURVE_COMP, OUT_MY_CURVE);</code>

Function	<code>void CurveComposition(IEngineCurve inCurve, int inCurveIndex, IEngineCurve compCurve, int compCurveIndex, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>IEngineCurve inCurve : input curve. int inCurveIndex : input curve index. IEngineCurve compCurve : composition curve. int compCurveIndex : composition curve index. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.</code>
Return	-
Description	Composite output curve[outCurveIndex] from an input curve[inCurveIndex] and a composite curve[compCurveIndex]. The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Operation.CurveComposition(IN_MY_CURVE, 9, IN_MY_CURVE_COMP, 0 OUT_MY_CURVE, 4);</code>

$$f(x[a]) = g_A(k(x[a]))$$

**5.15.3.26 ActDerive()**

Function	double ActDerive(IEngineCurve inCurve, IEngineCurve outCurve)
Parameter	IEngineCurve inCurve : input curve. IEngineCurve outCurve : output curve.
Return	double : Current derived value.
Description	Create the actual first derivative of an input curve[0] Y-value to an output curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	var derive = EngineLibrary.Operation.ActDerive(IN_MY_CURVE, OUT_MY_CURVE);

Function	double ActDerive(IEngineCurve inCurve, int inCurveIndex, IEngineCurve outCurve, int outCurveIndex)
Parameter	IEngineCurve inCurve : input curve. int inCurveIndex : input curve index. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.
Return	double : Current derived value.
Description	Create the actual first derivative of an input curve[inCurveIndex] Y-value to an output curve[outCurveIndex] Y-value. Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves only.
Example	var derive = EngineLibrary.Operation.ActDerive(IN_MY_CURVE, 2, OUT_MY_CURVE, 2);

**5.15.3.27 ActIntegrate()**

Function	<pre>double ActIntegrate(int key, IEngineCurve inCurve, IEngineCurve outCurve)</pre>
Parameter	<pre>int key : integration object key. IEngineCurve inCurve : input curve. IEngineCurve outCurve : output curve.</pre>
Return	<pre>double : Current integrated value.</pre>
Description	Create the actual integration of an input curve[0] Y-value to an output curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var derive = EngineLibrary.Operation.ActIntegrate(aKey, IN_MY_CURVE, OUT_MY_CURVE);</pre>

Function	<pre>double ActIntegrate(int key, IEngineCurve inCurve, int inCurveIndex, IEngineCurve outCurve, int outCurveIndex)</pre>
Parameter	<pre>int key : integration object key. IEngineCurve inCurve : input curve. int inCurveIndex : input curve index. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.</pre>
Return	<pre>double : Current integrated value.</pre>
Description	Create the actual integration of an input curve[inCurveIndex] Y-value to an output curve[outCurveIndex] Y-value. Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var derive = EngineLibrary.Operation.ActIntegrate(aKey, IN_MY_CURVE, 0, OUT_MY_CURVE, 8);</pre>

**5.15.3.28 ActRMA()**

Function	<code>double ActRMA(int key, IEngineCurve inCurve, IEngineCurve outCurve)</code>
Parameter	<code>int key</code> : running moving average (RMA) object key. <code>IEngineCurve inCurve</code> : input curve. <code>IEngineCurve outCurve</code> : output curve.
Return	<code>double</code> : Current RMA value.
Description	Create actual running moving average (RMA) of an input curve[0] Y-value to an output curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var rma = EngineLibrary.Operation.ActRMA(aKey, IN_MY_CURVE, OUT_MY_CURVE);</pre>

Function	<code>double ActRMA(int key, IEngineCurve inCurve, int inCurveIndex, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>int key</code> : running moving average (RMA) object key. <code>IEngineCurve inCurve</code> : input curve. <code>int inCurveIndex</code> : input curve index. <code>IEngineCurve outCurve</code> : output curve. <code>int outCurveIndex</code> : output curve index.
Return	<code>double</code> : Current RMA value.
Description	Create actual running moving average (RMA) of an input curve[inCurveIndex] Y-value to an output curve[outCurveIndex] Y-value. Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var rma = EngineLibrary.Operation.ActRMA(aKey, IN_MY_CURVE, 6, OUT_MY_CURVE, 3);</pre>

**5.15.3.29 ActRMAInt()**

Function	Int64 ActRMAInt(int key, IEngineCurve inCurve, IEngineCurve outCurve)
Parameter	int key : running moving average (RMA) integer object key. IEngineCurve inCurve : input curve. IEngineCurve outCurve : output curve.
Return	Int64 : Current RMA value.
Description	Create actual integer running moving average (RMA) of an input curve[0] Y-value to an output curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var rma = EngineLibrary.Operation.ActRMAInt(aKey, IN_MY_CURVE, OUT_MY_CURVE);</pre>

Function	Int64 ActRMAInt(int key, IEngineCurve inCurve, int inCurveIndex, IEngineCurve outCurve, int outCurveIndex)
Parameter	int key : running moving average (RMA) integer object key. IEngineCurve inCurve : input curve. int inCurveIndex : input curve index. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.
Return	Int64 : Current RMA value.
Description	Create actual integer running moving average (RMA) of an input curve[inCurveIndex] Y-value to an output curve[outCurveIndex] Y-value. Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var rma = EngineLibrary.Operation.ActRMAInt(aKey, IN_MY_CURVE, 7 OUT_MY_CURVE, 12);</pre>

**5.15.3.30 ActRMS()**

Function	<code>double ActRMS(int key, IEngineCurve inCurve, IEngineCurve outCurve)</code>
Parameter	<code>int key</code> : running root mean square (RMS) object key. <code>IEngineCurve inCurve</code> : input curve. <code>IEngineCurve outCurve</code> : output curve.
Return	<code>double</code> : Current RMA value.
Description	Create actual root mean square (RMS) of an input curve[0] Y-value to an output curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var rms = EngineLibrary.Operation.ActRMS(aKey, IN_MY_CURVE, OUT_MY_CURVE);</pre>

Function	<code>double ActRMS(int key, IEngineCurve inCurve, int inCurveIndex, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>int key</code> : root mean square (RMS) object key. <code>IEngineCurve inCurve</code> : input curve. <code>int inCurveIndex</code> : input curve index. <code>IEngineCurve outCurve</code> : output curve. <code>int outCurveIndex</code> : output curve index.
Return	<code>double</code> : Current RMS value.
Description	Create actual running moving average (RMS) of an input curve[inCurveIndex] Y-value to an output curve[outCurveIndex] Y-value. Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var rms = EngineLibrary.Operation.ActRMS(aKey, IN_MY_CURVE, 1 OUT_MY_CURVE, 4);</pre>

**5.15.3.31 ActAdd()**

Function	<code>double ActAdd(IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. IEngineCurve inCurveB : input curve B. IEngineCurve outCurve : output curve.</code>
Return	<code>double : Current added value.</code>
Description	Add actual input curve[0] A Y-value and actual input curve[0] B Y-value to output curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var add = EngineLibrary.Operation.ActAdd(IN_MY_CURVE_A, IN_MY_CURVE_B OUT_MY_CURVE);</pre>

Function	<code>double ActAdd(IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. int inCurveIndexA : input curve index A. IEngineCurve inCurveB : input curve B. int inCurveIndexB : input curve index B. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.</code>
Return	<code>double : Current added value.</code>
Description	Add actual input curve[inCurveIndexA] A Y-value and actual input curve[inCurveIndexB] B Y-value to output curve[outCurveIndex] Y-value. Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var add = EngineLibrary.Operation.ActAdd(IN_MY_CURVE_A, 4, IN_MY_CURVE_B, 9 OUT_MY_CURVE);</pre>

**5.15.3.32 ActAddInt()**

Function	Int64 ActAddInt(IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)
Parameter	IEngineCurve inCurveA : input curve A. IEngineCurve inCurveB : input curve B. IEngineCurve outCurve : output curve.
Return	Int64 : Current added value.
Description	Add actual integer input curve[0] A Y-value and actual integer input curve[0] B Y-value to integer output curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var add = EngineLibrary.Operation.ActAddInt(IN_MY_CURVE_A, IN_MY_CURVE_B OUT_MY_CURVE);</pre>

Function	Int64 ActAdd(IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)
Parameter	IEngineCurve inCurveA : input curve A. int inCurveIndexA : input curve index A. IEngineCurve inCurveB : input curve B. int inCurveIndexB : input curve index B. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.
Return	Int64 : Current added value.
Description	Add actual integer input curve[inCurveIndexA] A Y-value and actual integer input curve[inCurveIndexB] B Y-value to integer output curve[outCurveIndex] Y-value. Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var add = EngineLibrary.Operation.ActAddInt(IN_MY_CURVE_A, 4, IN_MY_CURVE_B, 9 OUT_MY_CURVE);</pre>

**5.15.3.33 ActSum()**

Function	double ActSum(IEngineCurve inCurve, IEngineCurve outCurve)
Parameter	IEngineCurve inCurve : input curve. IEngineCurve outCurve : output curve.
Return	double : Current summarized value.
Description	Summarize all actual input curve[0...n] Y-values to output curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	var sum = EngineLibrary.Operation.ActSum(IN_MY_CURVE OUT_MY_CURVE);

Function	double ActSum(IEngineCurve inCurve, IEngineCurve outCurve, int outCurveIndex)
Parameter	IEngineCurve inCurve : input curve. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.
Return	double : Current summarized value.
Description	Summarize all actual input curve[0...n] Y-values to output curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	var sum = EngineLibrary.Operation.ActSum(IN_MY_CURVE OUT_MY_CURVE, 4);

**5.15.3.34 ActSumInt()**

Function	Int64 ActSumInt(IEngineCurve inCurve, IEngineCurve outCurve)
Parameter	IEngineCurve inCurve : input curve. IEngineCurve outCurve : output curve.
Return	Int64 : Current summarized value.
Description	Summarize all actual integer input curve[0..n] Y-values to integer output curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var sum = EngineLibrary.Operation.ActSumInt(IN_MY_CURVE OUT_MY_CURVE);</pre>

Function	Int64 ActSumInt(IEngineCurve inCurve, IEngineCurve outCurve, int outCurveIndex)
Parameter	IEngineCurve inCurve : input curve. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.
Return	Int64 : Current summarized value.
Description	Summarize all actual integer input curve[0..n] Y-values to integer output curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var sum = EngineLibrary.Operation.ActSumInt(IN_MY_CURVE OUT_MY_CURVE, 4);</pre>

**5.15.3.35 ActDiff()**

Function	<code>double ActDiff(IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. IEngineCurve inCurveB : input curve B. IEngineCurve outCurve : output curve.</code>
Return	<code>double : Current difference value.</code>
Description	Minus actual input curve[0] B Y-value from actual input curve[0] A Y-value to output curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var diff = EngineLibrary.Operation.ActDiff(IN_MY_CURVE_A, IN_MY_CURVE_B, OUT_MY_CURVE);</pre>

Function	<code>double ActDiff(IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. int inCurveIndexA : input curve index A. IEngineCurve inCurveB : input curve B. int inCurveIndexB : input curve index B. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.</code>
Return	<code>double : Current difference value.</code>
Description	Minus actual input curve[inCurveIndexB] B Y-value from actual input curve[inCurveIndexA] A Y-value to output curve[outCurveIndex] Y-value. Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var diff = EngineLibrary.Operation.ActDiff(IN_MY_CURVE_A, 4, IN_MY_CURVE_B, 9 OUT_MY_CURVE);</pre>

**5.15.3.36 ActDiffInt()**

Function	<code>Int64 ActDiffInt(IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. IEngineCurve inCurveB : input curve B. IEngineCurve outCurve : output curve.</code>
Return	<code>Int64 : Current integer difference value.</code>
Description	Minus actual integer input curve[0] B Y-value from actual integer input curve[0] A Y-value to integer output curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var diff = EngineLibrary.Operation.ActDiffInt(IN_MY_CURVE_A, IN_MY_CURVE_B OUT_MY_CURVE);</pre>

Function	<code>double ActDiffInt(IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. int inCurveIndexA : input curve index A. IEngineCurve inCurveB : input curve B. int inCurveIndexB : input curve index B. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.</code>
Return	<code>Int64 : Current integer difference value.</code>
Description	Minus actual integer input curve[inCurveIndexB] B Y-value from actual integer input curve[inCurveIndexA] A Y-value to integer output curve[outCurveIndex] Y-value. Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var diff = EngineLibrary.Operation.ActDiffInt(IN_MY_CURVE_A, 4, IN_MY_CURVE_B, 9 OUT_MY_CURVE);</pre>

**5.15.3.37 ActMul()**

Function	<code>double ActMul(IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. IEngineCurve inCurveB : input curve B. IEngineCurve outCurve : output curve.</code>
Return	<code>double : Current multiplied value.</code>
Description	Multiply actual input curve[0] A Y-value and actual input curve[0] B Y-value to output curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var mul = EngineLibrary.Operation.ActMul(IN_MY_CURVE_A, IN_MY_CURVE_B OUT_MY_CURVE);</pre>

Function	<code>double ActMul(IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. int inCurveIndexA : input curve index A. IEngineCurve inCurveB : input curve B. int inCurveIndexB : input curve index B. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.</code>
Return	<code>double : Current multiplied value.</code>
Description	Multiply actual input curve[inCurveIndexA] A Y-value and actual input curve[inCurveIndexB] B Y-value to output curve[outCurveIndex] Y-value. Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var mul = EngineLibrary.Operation.ActMul(IN_MY_CURVE_A, 4, IN_MY_CURVE_B, 9 OUT_MY_CURVE);</pre>

**5.15.3.38 ActMulInt()**

Function	Int64 ActMulInt(IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)
Parameter	IEngineCurve inCurveA : input curve A. IEngineCurve inCurveB : input curve B. IEngineCurve outCurve : output curve.
Return	Int64 : Current multiplied integer value.
Description	Multiply actual integer input curve[0] A Y-value and actual integer input curve[0] B Y-value to integer output curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var mul = EngineLibrary.Operation.ActMulInt(IN_MY_CURVE_A, IN_MY_CURVE_B OUT_MY_CURVE);</pre>

Function	Int64 ActMulInt(IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)
Parameter	IEngineCurve inCurveA : input curve A. int inCurveIndexA : input curve index A. IEngineCurve inCurveB : input curve B. int inCurveIndexB : input curve index B. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.
Return	Int64 : Current multiplied integer value.
Description	Multiply actual integer input curve[inCurveIndexA] A Y-value and actual integer input curve[inCurveIndexB] B Y-value to integer output curve[outCurveIndex] Y-value. Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var mul = EngineLibrary.Operation.ActMulInt(IN_MY_CURVE_A, 4, IN_MY_CURVE_B, 9 OUT_MY_CURVE);</pre>

**5.15.3.39 ActProduct()**

Function	double ActProduct(IEngineCurve inCurve, IEngineCurve outCurve)
Parameter	IEngineCurve inCurve : input curve. IEngineCurve outCurve : output curve.
Return	double : Current product value.
Description	Product of all actual input curve[0...n] Y-values to output curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	var prod = EngineLibrary.Operation.ActProduct(IN_MY_CURVE OUT_MY_CURVE);

Function	double ActProduct(IEngineCurve inCurve, IEngineCurve outCurve, int outCurveIndex)
Parameter	IEngineCurve inCurve : input curve. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.
Return	double : Current product value.
Description	Product of all actual input curve[0...n] Y-values to output curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	var prod = EngineLibrary.Operation.ActProduct(IN_MY_CURVE OUT_MY_CURVE, 4);

**5.15.3.40 ActProductInt()**

Function	Int64 ActProductInt(IEngineCurve inCurve, IEngineCurve outCurve)
Parameter	IEngineCurve inCurve : input curve. IEngineCurve outCurve : output curve.
Return	Int64 : Current product integer value.
Description	Product of all actual integer input curve[0...n] Y-values to integer output curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var prod = EngineLibrary.Operation.ActProductInt(IN_MY_CURVE OUT_MY_CURVE);</pre>

Function	Int64 ActProductInt(IEngineCurve inCurve, IEngineCurve outCurve, int outCurveIndex)
Parameter	IEngineCurve inCurve : input curve. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.
Return	Int64 : Current product integer value.
Description	Product of all actual integer input curve[0...n] Y-values to integer output curve[0] Y-value. Only the actual integer output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var prod = EngineLibrary.Operation.ActProductInt(IN_MY_CURVE OUT_MY_CURVE, 4);</pre>

**5.15.3.41 ActDiv()**

Function	<code>double ActDiv(IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. IEngineCurve inCurveB : input curve B. IEngineCurve outCurve : output curve.</code>
Return	<code>double : Current division value.</code>
Description	Divide actual input curve[0] B Y-value from actual input curve[0] A Y-value to output curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var div = EngineLibrary.Operation.ActDiv(IN_MY_CURVE_A, IN_MY_CURVE_B OUT_MY_CURVE);</pre>

Function	<code>double ActDiv(IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. int inCurveIndexA : input curve index A. IEngineCurve inCurveB : input curve B. int inCurveIndexB : input curve index B. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.</code>
Return	<code>double : Current division value.</code>
Description	Divide actual input curve[inCurveIndexB] B Y-value from actual input curve[inCurveIndexA] A Y-value to output curve[outCurveIndex] Y-value. Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var div = EngineLibrary.Operation.ActDiv(IN_MY_CURVE_A, 4, IN_MY_CURVE_B, 9 OUT_MY_CURVE);</pre>

**5.15.3.42 ActDivInt()**

Function	Int64 ActDivInt(IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)
Parameter	IEngineCurve inCurveA : input curve A. IEngineCurve inCurveB : input curve B. IEngineCurve outCurve : output curve.
Return	Int64 : Current division integer value.
Description	Divide actual integer input curve[0] B Y-value from actual integer input curve[0] A Y-value to integer output curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var div = EngineLibrary.Operation.ActDivInt(IN_MY_CURVE_A, IN_MY_CURVE_B OUT_MY_CURVE);</pre>

Function	Int64 ActDivInt(IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)
Parameter	IEngineCurve inCurveA : input curve A. int inCurveIndexA : input curve index A. IEngineCurve inCurveB : input curve B. int inCurveIndexB : input curve index B. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.
Return	Int64 : Current division integer value.
Description	Divide actual integer input curve[inCurveIndexB] B Y-value from actual integer input curve[inCurveIndexA] A Y-value to integer output curve[outCurveIndex] Y-value. Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var div = EngineLibrary.Operation.ActDivInt(IN_MY_CURVE_A, 4, IN_MY_CURVE_B, 9 OUT_MY_CURVE);</pre>

**5.15.3.43 ActFactorOffset()**

Function	<code>double ActFactorOffset(IEngineCurve inCurve, IEngineCurve outCurve, double factor, double offset)</code>
Parameter	<code>IEngineCurve incurve : input curve. IEngineCurve outCurve : output curve. double factor : factor value. double offset : offset value.</code>
Return	<code>double : Current calculated value.</code>
Description	Multiply a Factor and add an Offset from actual input curve[0] Y-value to actual output curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var faOf = EngineLibrary.Operation.ActFactorOffset(IN_MY_CURVE, OUT_MY_CURVE, 2.0, -10.0);</pre>

Function	<code>double ActFactorOffset(IEngineCurve inCurve, int inCurveIndex, IEngineCurve outCurve, int outCurveIndex, double factor, double offset)</code>
Parameter	<code>IEngineCurve inCurve : input curve. int inCurveIndex : input curve index. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index. double factor : factor value. double offset : offset value.</code>
Return	<code>double : Current calculated value.</code>
Description	Multiply a Factor and add an Offset from actual input curve[outCurveIndex] Y-value to actual output curve[outCurveIndex] Y-value. Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var faOf = EngineLibrary.Operation.ActFactorOffset(IN_MY_CURVE, 4, OUT_MY_CURVE, 6, 2.0, -10.0);</pre>

**5.15.3.44 ActFactorOffsetInt()**

Function	<code>Int64 ActFactorOffsetInt(IEngineCurve inCurve, IEngineCurve outCurve, Int64 factor, Int64 offset)</code>
Parameter	<code>IEngineCurve inCurve : input curve. IEngineCurve outCurve : output curve. Int64 factor : factor value. Int64 offset : offset value.</code>
Return	<code>Int64 : Current calculated value.</code>
Description	Multiply an integer Factor and add an integer Offset from actual integer input curve[0] Y-value to actual integer output curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var faOf = EngineLibrary.Operation.ActFactorOffsetInt(IN_MY_CURVE, OUT_MY_CURVE, 2, -10);</pre>

Function	<code>Int64 ActFactorOffset(IEngineCurve inCurve, int inCurveIndex, IEngineCurve outCurve, int outCurveIndex, Int64 factor, Int64 offset)</code>
Parameter	<code>IEngineCurve inCurve : input curve. int inCurveIndex : input curve index. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index. Int64 factor : factor value. Int64 offset : offset value.</code>
Return	<code>Int64 : Current calculated value.</code>
Description	Multiply an integer Factor and add an integer Offset from actual integer input curve[outCurveIndex] Y-value to actual integer output curve[outCurveIndex] Y-value. Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var faOf = EngineLibrary.Operation.ActFactorOffsetInt(IN_MY_CURVE, 4, OUT_MY_CURVE, 6, 2.0, -10.0);</pre>

**5.15.3.45 ActConvertDB()**

Function	<code>double ActConvertDB(IEngineCurve inCurve, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurve</code> : input curve. <code>IEngineCurve outCurve</code> : output curve.
Return	<code>double</code> : Current calculated value.
Description	Convert Y-values of <code>inCurve[0]</code> to dezibel (DB) Y-values of <code>outCurve[0]</code> . Only the actual output curve[0] Y-value is calculated. Use in <code>Loop()</code> function for system curves only.
Example	<code>var db = EngineLibrary.Operation.ActConvertDB(IN_MY_CURVE, OUT_MY_CURVE);</code>

Function	<code>double ActConvertDB(IEngineCurve inCurve, int inCurveIndex, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>IEngineCurve inCurve</code> : input curve. <code>int inCurveIndex</code> : input curve index. <code>IEngineCurve outCurve</code> : output curve. <code>int outCurveIndex</code> : output curve index.
Return	<code>double</code> : Current calculated value.
Description	Convert Y-values of <code>inCurve[inCurveIndex]</code> to dezibel (DB) Y-values of <code>outCurve[outCurveIndex]</code> . Only the actual output curve[outCurveIndex] Y-value is calculated. Use in <code>Loop()</code> function for system curves only.
Example	<code>var db = EngineLibrary.Operation.ActConvertDB(IN_MY_CURVE, 3, OUT_MY_CURVE, 0);</code>

**5.15.3.46 ActComposition()**

Function	<code>double ActComposition(IEngineCurve inCurve, IEngineCurve compCurve, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurve : input curve. IEngineCurve compCurve : composition curve. IEngineCurve outCurve : output curve.</code>
Return	<code>double : Current calculated value.</code>
Description	Composite actual output curve[0] Y-value from an actual input curve[0] Y-value and an user composite curve[0] Y-value. <code>output curve[0][X][Y] = input curve[0][X][comp curve[0][X][Y]]</code> Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var comp = EngineLibrary.Operation.ActComposition(IN_MY_CURVE, IN_MY_CURVE_COMP, OUT_MY_CURVE);</pre>

Function	<code>double ActComposition(IEngineCurve inCurve, int inCurveIndex, IEngineCurve compCurve, int compCurveIndex, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>IEngineCurve inCurve : input curve. int inCurveIndex : input curve index. IEngineCurve compCurve : composition curve. int compCurveIndex : composition curve index. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.</code>
Return	<code>double : Current calculated value.</code>
Description	Composite actual output curve[outCurveIndex] Y-value from an actual input curve[inCurveIndex] Y-value and an user composite curve[compCurveIndex] Y-value. <code>output curve[outCurveIndex][X][Y] = input curve[inCurveIndex][X][comp curve[compCurveIndex][X][Y]]</code> Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var comp = EngineLibrary.Operation.ActComposition(IN_MY_CURVE, 2, IN_MY_CURVE_COMP, 0, OUT_MY_CURVE, 7);</pre>



5.15.4 Signal

5.15.4.1 CurvePhase()

Function	<code>void CurvePhase(IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. IEngineCurve inCurveB : input curve B. IEngineCurve outCurve : output curve.</code>
Return	-
Description	Calculate the phase between input curve[0] A and input curve[0] B to output curve[0]. The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Signal.CurvePhase(IN_MY_CURVE_A, IN_MY_CURVE_B, OUT_MY_CURVE);</code>

Function	<code>void CurvePhase(IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. int inCurveIndexA : input curve index A. IEngineCurve inCurveB : input curve B. int inCurveIndexB : input curve index B. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.</code>
Return	-
Description	Calculate the phase between input curve[inCurveIndexA] A and input curve[inCurveIndexB] B to output curve[outCurveIndex]. The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Signal.CurvePhase(IN_MY_CURVE_A, 1, IN_MY_CURVE_B, 1, OUT_MY_CURVE, 0);</code>

$$f(x[a]) = \text{atan}_2(g_A(x[a]), g_B(x[a]))$$



5.15.4.2 CurveMagnitude()

Function	<code>void CurveMagnitude(IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. IEngineCurve inCurveB : input curve B. IEngineCurve outCurve : output curve.</code>
Return	-
Description	Calculate the magnitude from input curve[0] A and input curve[0] B to output curve[0]. The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Signal.CurveMagnitude(IN_MY_CURVE_A, IN_MY_CURVE_B, OUT_MY_CURVE);</code>

Function	<code>void CurveMagnitude(IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurveC, int outCurveIndexC)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. int inCurveIndexA : input curve index A. IEngineCurve inCurveB : input curve B. int inCurveIndexB : input curve index B. IEngineCurve outcurve : output curve. int outCurveIndex : output curve index.</code>
Return	-
Description	Calculate the magnitude from input curve[inCurveIndexA] A and input curve[inCurveIndexB] B to output curve[outCurveIndex]. The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Signal.CurveMagnitude(IN_MY_CURVE_A, 1, IN_MY_CURVE_B, 1, OUT_MY_CURVE, 0);</code>

$$f(x[a]) = \sqrt{g_A(x[a])^2 + g_B(x[a])^2}$$

**5.15.4.3 CurveIQBaseband()**

Function	void CurveIQBaseBand(IEngineCurve inCurve, IEngineCurve outCurveI, IEngineCurve outCurveQ, double freqBase, double phase, double gain)
Parameter	IEngineCurve inCurveA : input curve. IEngineCurve outCurveI : output curve InPhase. IEngineCurve outCurveQ : output curve Quadrature. double freqBase : baseband frequency [Hz] double phase : phase [rad] double gain : gain
Return	-
Description	Calculate an InPhase output curve[0] I and a Quadrature output curve[0] Q from input curve[0]. Both output curve[0] are calculated at once. Use in Init() or Post() functions.
Example	EngineLibrary.Signal.CurveIQBaseband(IN_MY_CURVE, OUT_MY_CURVE_I, OUT_MY_CURVE_Q, 1E6, 0.785, 10.0);

Function	void CurveIQBaseBand(IEngineCurve inCurve, int inCurveIndex, IEngineCurve outCurveI, int outCurveIndexI, IEngineCurve outCurveQ, int outCurveIndexQ, double freqBase, double phase, double gain)
Parameter	IEngineCurve inCurveA : input curve. int inCurveIndex : input curve index. IEngineCurve outCurveI : output curve InPhase. int outCurveIndexI : output curve index I. IEngineCurve outCurveQ : output curve Quadrature. int outCurveIndexQ : output curve index Q. double freqBase : baseband frequency [Hz] double phase : phase [rad] double gain : gain
Return	-
Description	Calculate an InPhase output curve[outCurveIndexI] I and a Quadrature output curve[outCurveIndexQ] Q from input curve[inCurveIndex]. Both output curve[0] are calculated at once. Use in Init() or Post() functions.
Example	EngineLibrary.Signal.CurveIQBaseband(IN_MY_CURVE, 4, OUT_MY_CURVE_I, 4, OUT_MY_CURVE_Q, 4, 1E6, 0.785, 10.0);

$$f_I(x[a]) = \sin(2\pi f_{base} * x[a] + phase) * g(x[a]) * gain$$

$$f_Q(x[a]) = \cos(2\pi f_{base} * x[a] + phase) * g(x[a]) * gain$$

**5.15.4.4 ActPhase()**

Function	<code>double ActPhase(IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. IEngineCurve inCurveB : input curve B. IEngineCurve outCurve : output curve.</code>
Return	<code>double : Current phase value.</code>
Description	Calculate the phase between input curve[0] A and input curve[0] B to output curve[0]. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var phase = EngineLibrary.Signal.ActPhase(IN_MY_CURVE_A, IN_MY_CURVE_B, OUT_MY_CURVE);</pre>

Function	<code>double ActPhase(IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. int inCurveIndexA : input curve index A. IEngineCurve inCurveB : input curve B. int inCurveIndexB : input curve index B. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.</code>
Return	<code>double : Current phase value.</code>
Description	Calculate the phase between input curve[inCurveIndexA] A and input curve[inCurveIndexB] B to output curve[outCurveIndex]. Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var phase = EngineLibrary.Signal.ActPhase(IN_MY_CURVE_A, 1, IN_MY_CURVE_B, 1, OUT_MY_CURVE, 0);</pre>



5.15.4.5 ActMagnitude()

Function	<code>double ActMagnitude(IEngineCurve inCurveA, IEngineCurve inCurveB, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. IEngineCurve inCurveB : input curve B. IEngineCurve outCurve : output curve.</code>
Return	<code>double : Current magnitude value.</code>
Description	Calculate the magnitude from input curve[0] A and input curve[0] B to output curve[0]. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var mag = EngineLibrary.Signal.ActMagnitude(IN_MY_CURVE_A, IN_MY_CURVE_B, OUT_MY_CURVE);</pre>

Function	<code>double ActMagnitude(IEngineCurve inCurveA, int inCurveIndexA, IEngineCurve inCurveB, int inCurveIndexB, IEngineCurve outCurveC, int outCurveIndexC)</code>
Parameter	<code>IEngineCurve inCurveA : input curve A. int inCurveIndexA : input curve index A. IEngineCurve inCurveB : input curve B. int inCurveIndexB : input curve index B. IEngineCurve outcurve : output curve. int outCurveIndex : output curve index.</code>
Return	<code>double : Current magnitude value.</code>
Description	Calculate the magnitude from input curve[inCurveIndexA] A and input curve[inCurveIndexB] B to output curve[outCurveIndex]. Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves only.
Example	<pre>var mag = EngineLibrary.Signal.ActMagnitude(IN_MY_CURVE_A, 1, IN_MY_CURVE_B, 1, OUT_MY_CURVE, 0);</pre>



5.15.4.6 ActIQBaseband()

Function	<pre>double ActIQBaseBand(IEngineCurve inCurve, IEngineCurve outCurveI, IEngineCurve outCurveQ, double freqBase, double phase, double gain)</pre>
Parameter	<pre>IEngineCurve inCurveA : input curve. IEngineCurve outCurveI : output curve InPhase. IEngineCurve outCurveQ : output curve Quadrature. double freqBase : baseband frequency [Hz] double phase : phase [rad] double gain : gain</pre>
Return	<pre>double : Current magnitude of IQ.</pre>
Description	Calculate an InPhase output curve[0] I and a Quadrature output curve[0] Q from input curve[0]. Only the actual output curve[0] I Y-value and actual output curve[0] Q Y-value are calculated. Use in Loop() function for system curves only.
Example	<pre>var mag = EngineLibrary.Signal.ActIQBaseband(IN_MY_CURVE, OUT_MY_CURVE_I, OUT_MY_CURVE_Q, 1E6, 0.785, 10.0);</pre>

Function	<pre>void CurveIQBaseBand(IEngineCurve inCurve, int inCurveIndex, IEngineCurve outCurveI, int outCurveIndexI, IEngineCurve outCurveQ, int outCurveIndexQ, double freqBase, double phase, double gain)</pre>
Parameter	<pre>IEngineCurve inCurveA : input curve. int inCurveIndex : input curve index. IEngineCurve outCurveI : output curve InPhase. int outCurveIndexI : output curve index I. IEngineCurve outCurveQ : output curve Quadrature. int outCurveIndexQ : output curve index Q. double freqBase : baseband frequency [Hz] double phase : phase [rad] double gain : gain</pre>
Return	<pre>double : Current magnitude of IQ.</pre>
Description	Calculate an InPhase output curve[outCurveIndexI] I and a Quadrature output curve[outCurveIndexQ] Q from input curve[inCurveIndex]. Only the actual output curve[outCurveIndexI] I Y-value and actual output curve[outCurveIndexQ] Q Y-value are calculated. Use in Loop() function for system curves only.
Example	<pre>var mag = EngineLibrary.Signal.ActIQBaseband(IN_MY_CURVE, 4, OUT_MY_CURVE_I, 4, OUT_MY_CURVE_Q, 4, 1E6, 0.785, 10.0);</pre>



5.15.5 Source

5.15.5.1 CreateSinus()

Function	<code>int CreateSinus(double amplitude, double offset, double step, Int64 dT1, Int64 dT2, Int64 dT3, Int64 phaseT, SourcePeriodicity periodicity</code>
Parameter	<code>double amplitude</code> : relative sinus amplitude. <code>double offset</code> : absolute sinus offset from zero. <code>double step</code> : time for one step ($\geq$ time tick). <code>Int64 dT1</code> : time steps for phase cutting at sinus start. <code>Int64 dT2</code> : time steps normal sinus duration. <code>Int64 dT3</code> : time steps for phase cutting at sinus end. <code>Int64 phaseT</code> : time steps for sinus start phase. <code>SourcePeriodicity periodicity</code> : kind of periodicity (normal, rectified, zero)
Return	<code>int</code> : sinus source object key.
Description	Create a sinus source object with different timing and signal parameters.
Example	<pre>var key = EngineLibrary.Source.CreateSinus(325.0, 0, 0.001, 0, 10, 0, 2, SourcePeriodicity.normal);</pre>

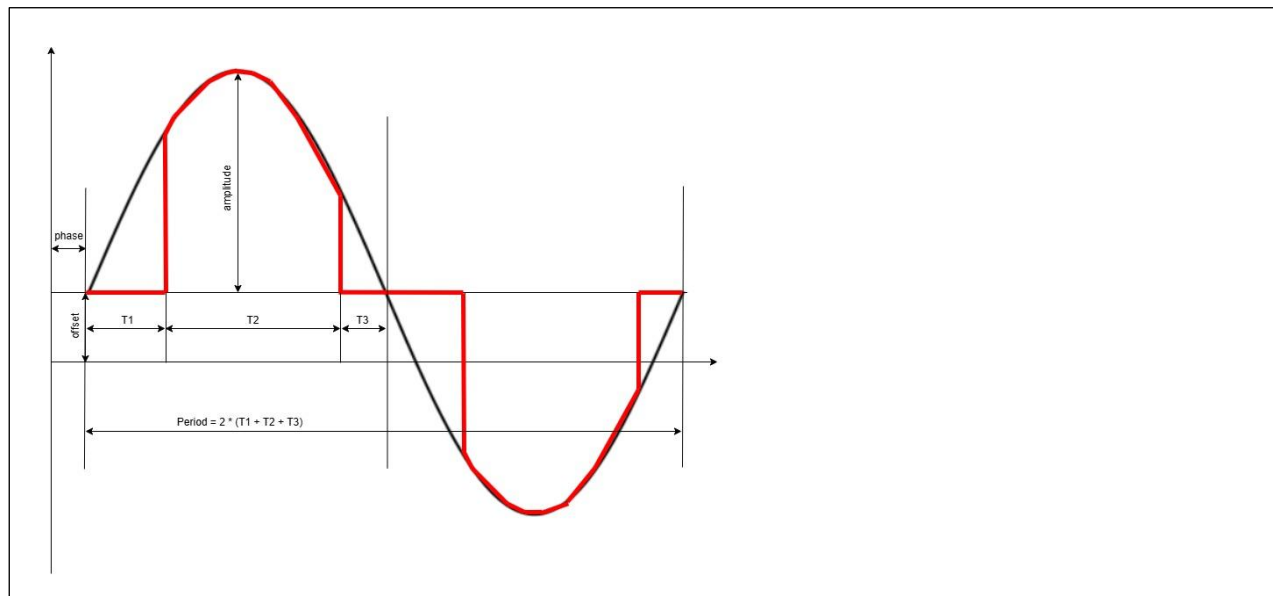


Figure 32 Sinus Source Definition.



5.15.5.2 CreateLinear()

Function	<pre>int CreateLinear(double amplitude, double offset, double step, Int64 dT1, Int64 dT2, Int64 dT3, Int64 dT4, Int64 dT5, Int64 phaseT, SourcePeriodicity periodicity)</pre>
Parameter	<pre>double amplitude : relative amplitude. double offset : absolute offset from zero. double step : time for one step (>= time tick). Int64 dT1 : time steps for zero at start. Int64 dT2 : time steps for linear increase. Int64 dT3 : time steps for amplitude duration. Int64 dT4 : time steps for linear decrease. Int64 dT5 : time steps for zero at end. Int64 phaseT : time steps for sinus start phase. SourcePeriodicity periodicity : kind of periodicity (normal, rectified, zero)</pre>
Return	<pre>int : linear source object key.</pre>
Description	Create a linear source object with different timing and signal parameters.
Example	<pre>var key = EngineLibrary.Source.CreateLinear(10.0, 0, 0.001, 0, 5, 10, 5, 0, 0, SourcePeriodicity.normal);</pre>

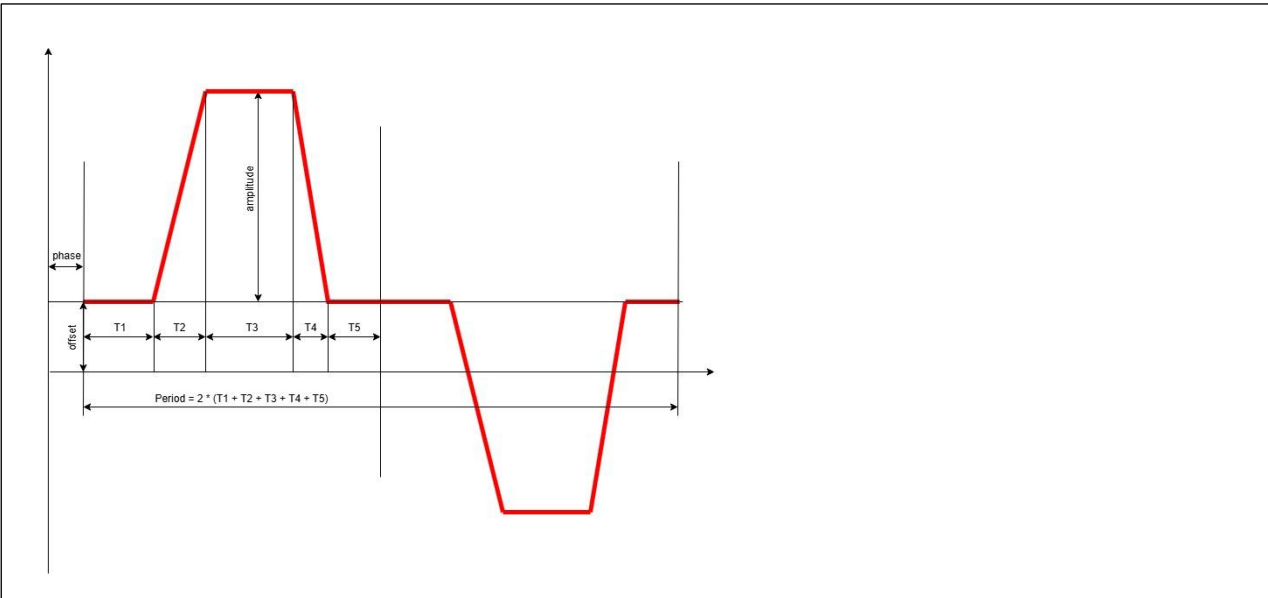


Figure 33 Linear Source Definition.

**5.15.5.3 CreateRandom()**

Function	<code>int CreateRandom(double amplitude, double offset, bool positive, bool negative</code>
Parameter	<code>double amplitude</code> : relative amplitude. <code>double offset</code> : absolute offset from zero. <code>bool positive</code> : True = positive values, False = no positive values. <code>bool negative</code> : True = negative values, False = no negative values.
Return	<code>int</code> : random source object key.
Description	Create a random source object with different signal parameters.
Example	<code>var key = EngineLibrary.Source.CreateLinear(50.0, -10.0, true, false);</code>



5.15.5.4 CurveSinus()

Function	<code>void CurveSinus(int key, IEngineCurve outCurve)</code>
Parameter	<code>int key</code> : created sinus object key. <code>IEngineCurve outCurve</code> : output curve.
Return	-
Description	Calculate sinus source to output curve[0]. The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Source.CurveSinus(aKey, OUT_MY_CURVE);</code>

Function	<code>void CurveSinus(int key, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>int key</code> : created sinus object key. <code>IEngineCurve outCurve</code> : output curve. <code>int outCurveIndex</code> : output curve index.
Return	-
Description	Calculate sinus source to output curve[outCurveIndex]. The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Source.CurveSinus(aKey, OUT_MY_CURVE, 3);</code>



5.15.5.5 CurveLinear()

Function	<code>void CurveLinear(int key, IEngineCurve outCurve)</code>
Parameter	<code>int key</code> : created linear object key. <code>IEngineCurve outCurve</code> : output curve.
Return	-
Description	Calculate linear source to output curve[0]. The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Source.CurveLinear(aKey, OUT_MY_CURVE);</code>

Function	<code>void CurveLinear(int key, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>int key</code> : created linear object key. <code>IEngineCurve outCurve</code> : output curve. <code>int outCurveIndex</code> : output curve index.
Return	-
Description	Calculate linear source to output curve[outCurveIndex]. The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Source.CurveLinear(aKey, OUT_MY_CURVE, 3);</code>



5.15.5.6 CurveRandom()

Function	<code>void CurveRandom(int key, IEngineCurve outCurve)</code>
Parameter	<code>int key</code> : created random object key. <code>IEngineCurve outCurve</code> : output curve.
Return	-
Description	Calculate random source to output curve[0]. The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Source.CurveRandom(aKey, OUT_MY_CURVE);</code>

Function	<code>void CurveRandom(int key, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>int key</code> : created random object key. <code>IEngineCurve outCurve</code> : output curve. <code>int outCurveIndex</code> : output curve index.
Return	-
Description	Calculate random source to output curve[outCurveIndex]. The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Source.CurveRandom(aKey, OUT_MY_CURVE, 3);</code>

**5.15.5.7 ActSinus()**

Function	double ActSinus(int key, IEngineCurve outCurve)
Parameter	int key : created sinus object key. IEngineCurve outCurve : output curve.
Return	double : Current sinus source Y-value.
Description	Calculate actual sinus source to output curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only. .
Example	var sin = EngineLibrary.Source.ActSinus(aKey, OUT_MY_CURVE);

Function	double ActSinus(int key, IEngineCurve outCurve, int outCurveIndex)
Parameter	int key : created sinus object key. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.
Return	double : Current sinus source Y-value.
Description	Calculate actual sinus source to output curve[outCurveIndex] Y-value. Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves only.
Example	var sin = EngineLibrary.Source.CurveSinus(aKey, OUT_MY_CURVE, 3);

**5.15.5.8 ActLinear()**

Function	double ActLinear(int key, IEngineCurve outCurve)
Parameter	int key : created linear object key. IEngineCurve outCurve : output curve.
Return	double : Current linear source Y-value.
Description	Calculate actual linear source to output curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only. .
Example	var lin = EngineLibrary.Source.ActLinear(aKey, OUT_MY_CURVE);

Function	double ActLinear(int key, IEngineCurve outCurve, int outCurveIndex)
Parameter	int key : created linear object key. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.
Return	double : Current linear source Y-value.
Description	Calculate actual linear source to output curve[outCurveIndex] Y-value. Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves only.
Example	var lin = EngineLibrary.Source.CurveLinear(aKey, OUT_MY_CURVE, 3);

**5.15.5.9 ActRandom()**

Function	double ActRandom(int key, IEngineCurve outCurve)
Parameter	int key : created random object key. IEngineCurve outCurve : output curve.
Return	double : Current random source Y-value.
Description	Calculate actual random source to output curve[0] Y-value. Only the actual output curve[0] Y-value is calculated. Use in Loop() function for system curves only. .
Example	var ran = EngineLibrary.Source.ActRandom(aKey, OUT_MY_CURVE);

Function	double ActRandom(int key, IEngineCurve outCurve, int outCurveIndex)
Parameter	int key : created random object key. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index.
Return	double : Current random source Y-value.
Description	Calculate actual random source to output curve[outCurveIndex] Y-value. Only the actual output curve[outCurveIndex] Y-value is calculated. Use in Loop() function for system curves only.
Example	var ran = EngineLibrary.Source.CurveRandom(aKey, OUT_MY_CURVE, 3);



5.15.6 Spectrum

5.15.6.1 CurveFourier()

Function	<code>void CurveFourier(IEngineCurve inCurve, IEngineCurve outCurve, int waves)</code>
Parameter	<code>IEngineCurve inCurve : input curve. IEngineCurve outCurve : output curve. int waves : number of transformed periodic signals</code>
Return	-
Description	Calculate Fourier transformed output curve[0] according to input curve[0]. The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Spectrum.CurveFourier(IN_MY_CURVE, OUT_MY_CURVE, 50);</code>

Function	<code>void CurveFourier(IEngineCurve inCurve, int inCurveIndex, IEngineCurve outCurve, int outCurveIndex, int waves)</code>
Parameter	<code>IEngineCurve inCurve : input curve. int inCurveIndex : input curve index. IEngineCurve outCurve : output curve. int outCurveIndex : output curve index. int waves : number of transformed periodic signals</code>
Return	-
Description	Calculate Fourier transformed output curve[outCurveIndex] according to input curve[inCurveIndex]. The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Spectrum.CurveFourier(IN_MY_CURVE, 7 OUT_MY_CURVE, 7, 50);</code>

$$x_{approx}[n] = \frac{1}{N} \sum_{k=0}^{M-1} X[k] * e^{j\frac{2\pi}{N}kn}$$



5.15.6.2 CurveFFT()

Function	<code>void CurveFFT(IEngineCurve inCurve, IEngineCurve outCurve)</code>
Parameter	<code>IEngineCurve inCurve</code> : input curve. <code>IEngineCurve outCurve</code> : output curve.
Return	-
Description	Calculate FFT (Fast Fourier Transformation) spectrum output curve[0] according to input curve[0]. The entire output curve[0] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Spectrum.CurveFFT(IN_MY_CURVE, OUT_MY_CURVE);</code>

Function	<code>void CurveFFT(IEngineCurve inCurve, int inCurveIndex, IEngineCurve outCurve, int outCurveIndex)</code>
Parameter	<code>IEngineCurve inCurve</code> : input curve. <code>int inCurveIndex</code> : input curve index. <code>IEngineCurve outCurve</code> : output curve. <code>int outCurveIndex</code> : output curve index.
Return	-
Description	Calculate FFT (Fast Fourier Transformation) spectrum output curve[outCurveIndex] according to input curve[inCurveIndex]. The entire output curve[outCurveIndex] is calculated at once. Use in Init() or Post() functions.
Example	<code>EngineLibrary.Spectrum.CurveFFT(IN_MY_CURVE, 7 OUT_MY_CURVE, 7);</code>

$$X[k] = \frac{1}{N} \sum_{n=0}^{N-1} x[n] * e^{-j\frac{2\pi}{N}kn}$$



5.16 Engine Track

5.16.1 Data Structure

Engine Track Data Structure

```
[StructLayout(LayoutKind.Sequential, Pack = 1)]
public struct SEngineTrack
{
    //
    // Has to be similar to the native struct declaration
    //
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 256)]
    public string text;
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 1024)]
    public byte[] byteArray;
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 256)]
    public double[] dArray;
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 256)]
    public Int64[] iArray;
    public double dValueA;
    public double dValueB;
    public Int64 iValueA;
    public Int64 iValueB;
}
```



5.16.2 Master Element

5.16.2.1 WriteGlobalEvent()

Function	<code>void WriteGlobalEvent(string msg, TrackMsgType msgType)</code>
Parameter	<code>string msg</code> : message text. <code>TrackMsgType msgType</code> : message type: Information, Warning, Error.
Return	-
Description	Send a message text to the global output event window.
Example	<code>EngineTrack.WriteGlobalEvent("Voltage source is switching on.", TrackMsgType.Information);</code>

5.16.2.2 WriteLocalEvent()

Function	<code>void WriteLocalEvent(string msg, int keyElement, TrackMsgType msgType)</code>
Parameter	<code>string msg</code> : message text. <code>int keyElement</code> : element key number ID <code>TrackMsgType msgType</code> : message type: Information, Warning, Error.
Return	-
Description	Send a message text to a local event window.
Example	<code>EngineTrack.WriteGlobalEvent("Voltage source is switching on.", 1, TrackMsgType.Information);</code>

5.16.2.3 Write()

Function	<code>void Write(int trackNumber, SEngineTrack trackData)</code>
Parameter	<code>int trackNumber</code> : track number defined in track settings. <code>SEngineTrack trackData</code> : track data object to write.
Return	-
Description	Send defined track data on the trackNumber channel.
Example	<code>EngineTrack.Write(100009, trackData);</code>

5.16.2.4 Read()

Function	<code>void Read(int trackNumber, ref SEngineTrack trackData)</code>
Parameter	<code>int trackNumber</code> : track number defined in track settings. <code>ref SEngineTrack trackData</code> : reference to track data object.
Return	-
Description	Send defined track data on the trackNumber channel.
Example	<code>EngineTrack.Read(100009, ref trackData);</code>



5.16.3 Master Element to Assembly

5.16.3.1 Transmit()

Function	<code>void Transmit(int trackNumber, SEngineTrack trackData)</code>
Parameter	<code>int trackNumber</code> : track number defined in track settings. <code>SEngineTrack trackData</code> : track data object to write.
Return	-
Description	Send defined track data on the trackNumber channel from an Element to an embedded assembly slave.
Example	<code>EngineTrack.Transmit(100012, trackData);</code>

5.16.3.2 Receive()

Function	<code>void Receive(int trackNumber, ref SEngineTrack trackData)</code>
Parameter	<code>int trackNumber</code> : track number defined in track settings. <code>ref SEngineTrack trackData</code> : reference to track data object.
Return	-
Description	Get embedded assembly slave defined track data on the trackNumber channel to Element.
Example	<code>EngineTrack.Receive(100012, ref trackData);</code>



5.16.4 Slave Assembly Information

Use following functions and variable for your track slave assembly.

5.16.4.1 Subscribe()

Function	<code>void Subscribe(IEngineTrackSlave slave)</code>
Parameter	<code>IEngineTrackSlave slave</code> : .NET Framework assembly which derived from <code>EngineTrackSalve</code>
Return	-
Description	Subscribe your assembly to connect with BraceCalculation Element.
Example	<code>EngineTrack.Subscribe(this);</code>

Public Variables	<code>string BraceCalcVersion</code> : get current BraceCalculation version string. <code>string CreationTime</code> : get time of compiled BraceCalculation user code. <code>bool Running</code> : flag if calculation is running. <code>bool Paused</code> : flag if calculation is paused. <code>bool Stepped</code> : flag if calculation is stepped. <code>double TickTime</code> : get used tick time. <code>double SeqTime</code> : get used sequence time. <code>double CurSeqTime</code> : get current processed sequence time. <code>Int64 SeqTicks</code> : get used ticks for one sequence. <code>Int64 CurSeqTicks</code> : get current processed sequence ticks. <code>Int64 TotalTicks</code> : get total ticks. <code>Int64 CurTotalTicks</code> : get total processed ticks. <code>double TotalTime</code> : get total calculation time. <code>double CurTotalTime</code> : get processed total calculation time. <code>double Sequences</code> : get calculation sequences. <code>Int64 CurSequences</code> : get number of processed complete sequences.
-------------------------	--

5.16.4.2 WriteGlobalEvent()

See master function.

5.16.4.3 WriteLocalEvent()

See master function.



5.16.5 Slave Assembly to Element

5.16.5.1 Transmit()

Function	<code>void Transmit(int trackNumber, ref SEngineTrack trackData, Int64 curTicks, double curTime)</code>
Parameter	<code>int trackNumber</code> : track number defined in track settings. <code>SEngineTrack trackData</code> : track data object reference to transmit to Element. <code>Int64 curTicks</code> : current processed time ticks. <code>double curTime</code> : current processed time.
Return	-
Description	Define your transmit function into your assembly slave class.
Example	<pre>EngineTrack.Transmit(int trackNumber, ref SEngineTrack trackData, Int64 curTicks, double curTime) { // // Put your code and set the trackData for an Element // if(trackNumber == 100009) { trackData.dValueA = 3.1415; } }</pre>

5.16.5.2 Receive()

Function	<code>void Receive(int trackNumber, SEngineTrack trackData, Int64 curTicks, double curTime)</code>
Parameter	<code>int trackNumber</code> : track number defined in track settings. <code>SEngineTrack trackData</code> : track data object to transmit to Element. <code>Int64 curTicks</code> : current processed time ticks. <code>double curTime</code> : current processed time.
Return	-
Description	Define your transmit function into your assembly slave class.
Example	<pre>EngineTrack.Receive(int trackNumber, ref SEngineTrack trackData, Int64 curTicks, double curTime) { // // Put your code and get the trackData for assembly usage. // if(trackNumber == 100019) { myValue = trackData.dValueA; } }</pre>

6 Examples

6.1 Series Oscillation Circuit

Example file for simple implementation: SerialRLC_simple.xcalc
 Example file for extended implementation: SerialRLC_extended.xcalc

6.1.1 Overview

Here we describe the mathematical calculations and the simulation of a series resonant circuit using **BraceCalculation**.

The signal flow corresponds to that shown on the title page of this document.

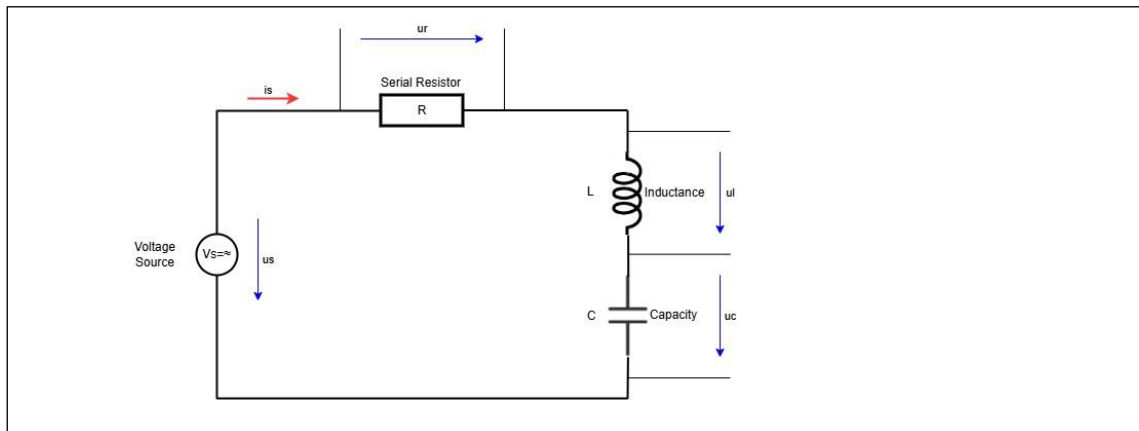


Figure 34 Serial oscillation circuit

Based on the laws of knots and meshes:

$$us = ur + ul + uc$$

$$ur = R \cdot is$$

$$ur = R \cdot is = us - ul - uc$$

$$uc = us - ur - ul = us - R \cdot is - ul$$

$$ul = us - ur - uc = us - R \cdot is - uc$$

$$ul(t) = L \cdot \frac{dis(t)}{dt} \Rightarrow \text{Laplace: } ul(s) = Ls \cdot I(s)$$

$$il(t) = \frac{1}{L} \cdot \int ul(t) dt \Rightarrow \text{Laplace: } il(s) = \frac{1}{Ls} \cdot U(s)$$

$$uc(t) = \frac{1}{C} \cdot \int is(t) dt \Rightarrow \text{Laplace: } uc(s) = \frac{1}{Cs} \cdot I(s)$$

We are also interested in the ratio of voltage uc to voltage us .

$$G(t) = \frac{uc(t)}{us(t)} = \frac{uc(t)}{ur(t) + ul(t) + uc(t)} \Rightarrow \text{Laplace: } G(s) = \frac{\frac{1}{Cs} \cdot I(s)}{R \cdot I(s) + Ls \cdot I(s) + \frac{1}{Cs} \cdot I(s)}$$

Now we can reduce the current and simplify:

$$G(s) = \frac{1}{RCs + LCs^2 + 1}$$

6.1.2 BraceCalculation Simple Implementation

We have created the **BraceCalculation** file SerialRLC_simple.xcalc.

According to the laws of knots and meshes we draw following schematic:

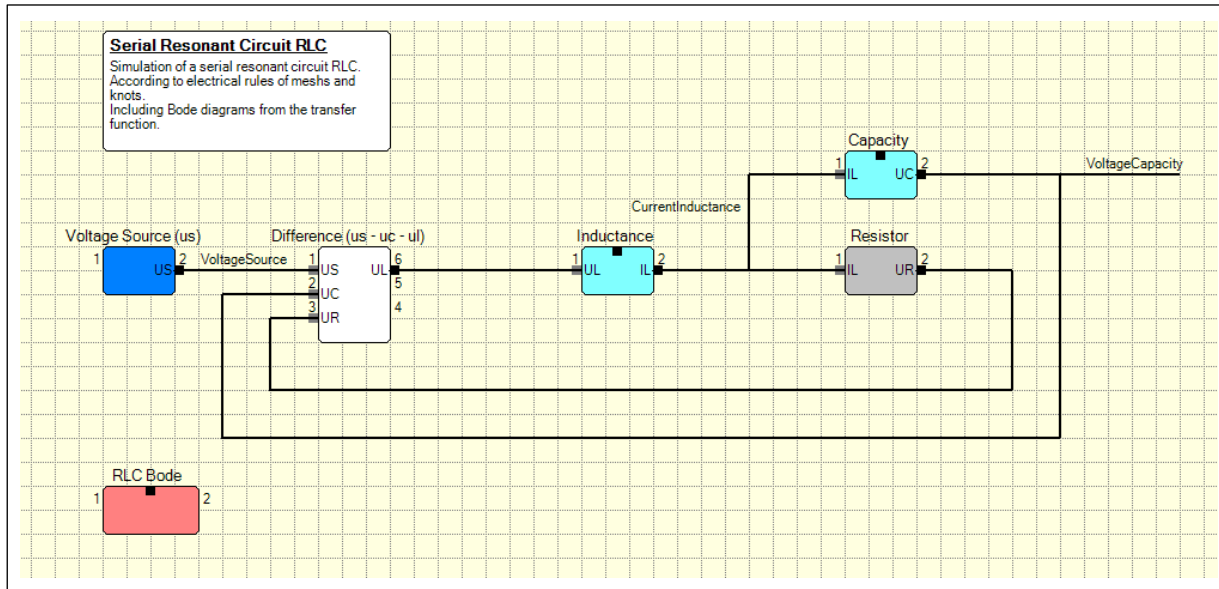


Figure 35 Schematic for a serial oscillation circuit

Timing

- One Tick Duration: 5E-8 second
- Sequence Duration: 1000000 ticks
- Number of Sequences: 1

Elements

Name	Pins	Curves	Remarks
VoltageSource	Output US	VoltageSource (SystemTime)	Voltage switches on after 10% simulation time.
Difference	Input US Input UC Input UR Output UI	VoltageSource (SystemTime) VoltageCapacity (SystemTime) VoltageResistor (SystemTime) VoltageInductance (SystemTime)	Voltage differences according to mesh.
Inductance	Input UL Output IL	VoltageInductance (SystemTime) CurrentInductance (SystemTime)	Current is an integral from the voltage.
Capacity	Input IL Output UC	CurrentInductance (SystemTime) VoltageCapacity (SystemTime)	Voltage is an integral from the current.
Resistor	Input IL Output UR	CurrentInductance (SystemTime) VoltageResistor (SystemTime)	Voltage is linear from the current.
RLC Bode	Internal AR Internal PR	AmplitudeResponse (User def, 1..1000Hz log) PhaseResponse (User def, 1..1000Hz log)	Static calculated transfer function.



Voltage Source (us) User Code

```
private double us = 0;

public void Init(Int64 maxTicks, double tickTime)
{
}

public void Loop(Int64 maxTicks, Int64 curTicks, double curTime, double tickTime)
{
    //
    // We create a step from 0 to 5.0V after 10% time
    //
    if(curTicks < (maxTicks / 10))
    {
        us = 0;
    }
    else
    {
        us = 5.0;
    }

    OUT_VoltageSource.SetYDouble(us);
}

public void Post(Int64 maxTicks, double tickTime)
{
}
```

Difference (us – uc – ul) User Code

```
public void Init(Int64 maxTicks, double tickTime)
{
}

public void Loop(Int64 maxTicks, Int64 curTicks, double curTime, double tickTime)
{
    //
    // We create the difference: ur = us - ul - uc
    //
    var us = IN_VoltageSource.GetYDouble();
    var ur = IN_VoltageResistor.GetYDouble();
    var uc = IN_VoltageCapacity.GetYDouble();

    OUT_VoltageInductance.SetYDouble(us - ur - uc);
}

public void Post(Int64 maxTicks, double tickTime)
{
}
```



Inductance User Code

```
private double L = 2.0; // take care your inductance is not too small
private int key = 0;

public void Init(Int64 maxTicks, double tickTime)
{
    key = EngineLibrary.Operation.CreateIntegration(0);
}

public void Loop(Int64 maxTicks, Int64 curTicks, double curTime, double tickTime)
{
    //
    // We create the inductance current
    //
    var volt = INTERNAL_Charge.GetYDouble();
    OUT_CurrentInductance.SetYDouble(volt / L);

    // Integrate to output curve
    EngineLibrary.Operation.ActIntegrate(key, IN_VoltageInductance, INTERNAL_Charge);
}

public void Post(Int64 maxTicks, double tickTime)
{
}
```

Capacity User Code

```
private double C = 1E-6;
private int key = 0;

public void Init(Int64 maxTicks, double tickTime)
{
    key = EngineLibrary.Operation.CreateIntegration(0);
}

public void Loop(Int64 maxTicks, Int64 curTicks, double curTime, double tickTime)
{
    // Coulomb to internal curve
    var current = IN_CurrentInductance.GetYDouble();
    INTERNAL_Coulomb.SetYDouble(current / C);

    // Integrate to output curve
    EngineLibrary.Operation.ActIntegrate(key, INTERNAL_Coulomb, OUT_VoltageCapacity);
}

public void Post(Int64 maxTicks, double tickTime)
{
}
```



Resistor User Code

```
private double R = 1000.0;

public void Init(Int64 maxTicks, double tickTime)
{
}

public void Loop(Int64 maxTicks, Int64 curTicks, double curTime, double tickTime)
{
    //
    // We create the resistor current
    //
    var cur = IN_CurrentInductance.GetYDouble();

    OUT_VoltageResistor.SetYDouble(cur * R);
}

public void Post(Int64 maxTicks, double tickTime)
{
}
```

RLC Bode User Code

```
public void Init(Int64 maxTicks, double tickTime)
{
    //
    // RLC
    //
    var R      = 1000;
    var L      = 2.0;
    var C      = 1E-6;
    var coA    = new double[] {1.0};
    var coB    = new double[] {1.0, R * C, L * C};
    var rlcBode = EngineLibrary.Filter.CreateAnalogPolynom(coA, coB);

    EngineLibrary.Filter.CurveFilterAmplitudeResponse(rlcBode, INTERNAL_AmplitudeResponse);
    EngineLibrary.Operation.CurveConvertDB(INTERNAL_AmplitudeResponse,
                                           INTERNAL_AmplitudeResponse);

    EngineLibrary.Filter.CurveFilterPhaseResponse(rlcBode, INTERNAL_PhaseResponse);
}

public void Loop(Int64 maxTicks, Int64 curTicks, double curTime, double tickTime)
{
}

public void Post(Int64 maxTicks, double tickTime)
{
}
```

6.1.3 BraceCalculation Simple Simulation

After Build and Start the following curves are visible in the different scopes.

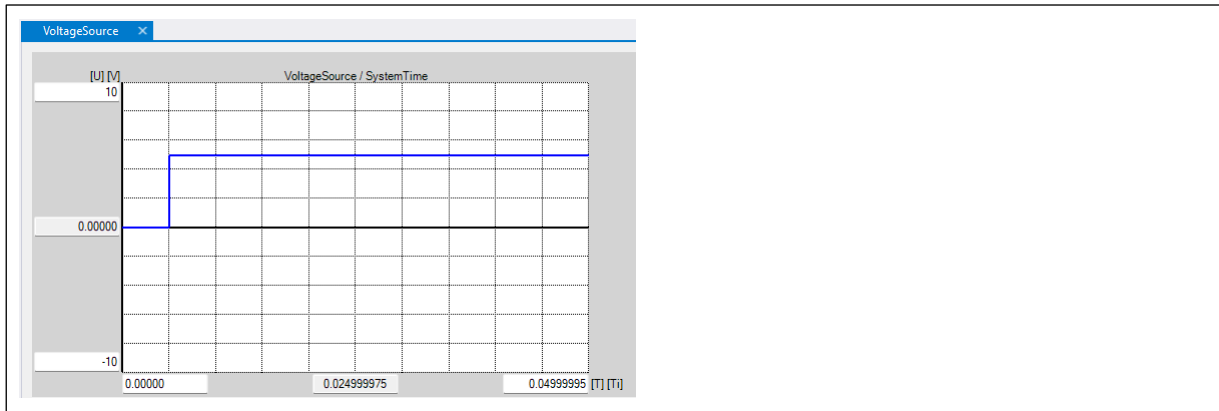


Figure 36 Voltage Source

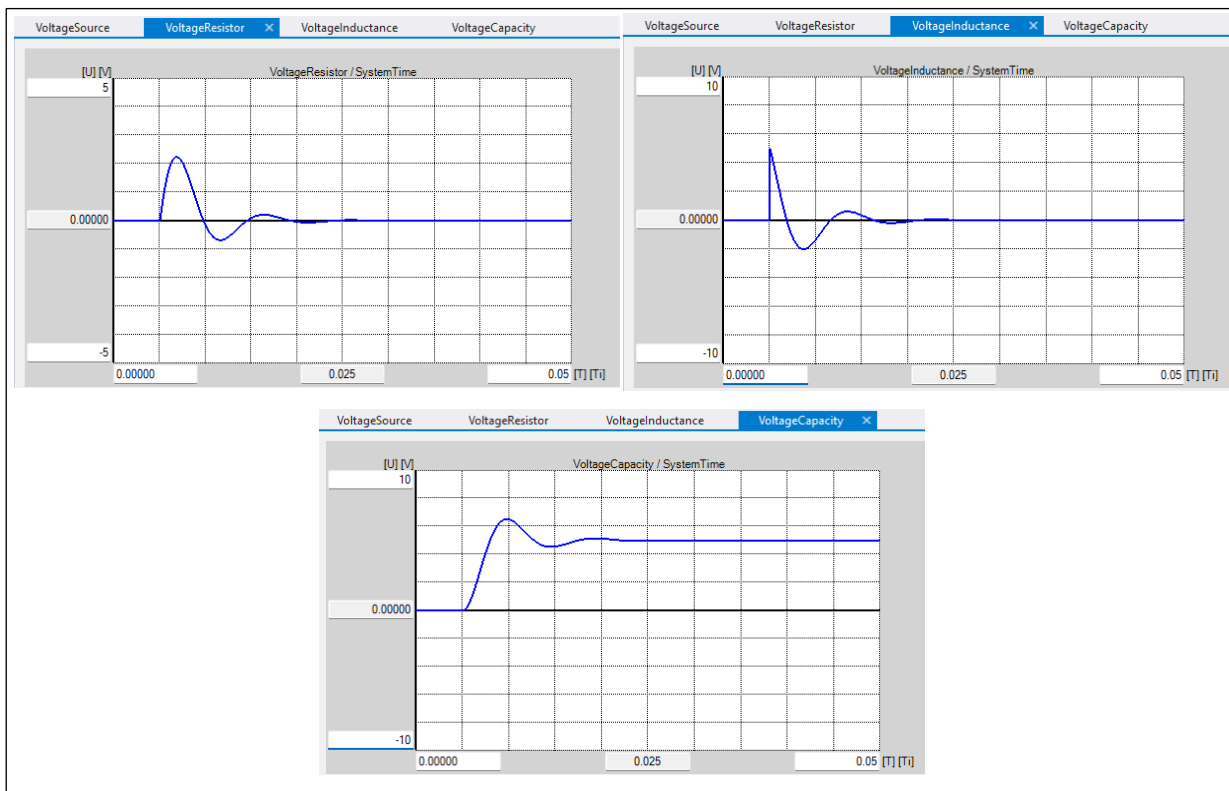


Figure 37 Voltages RLC

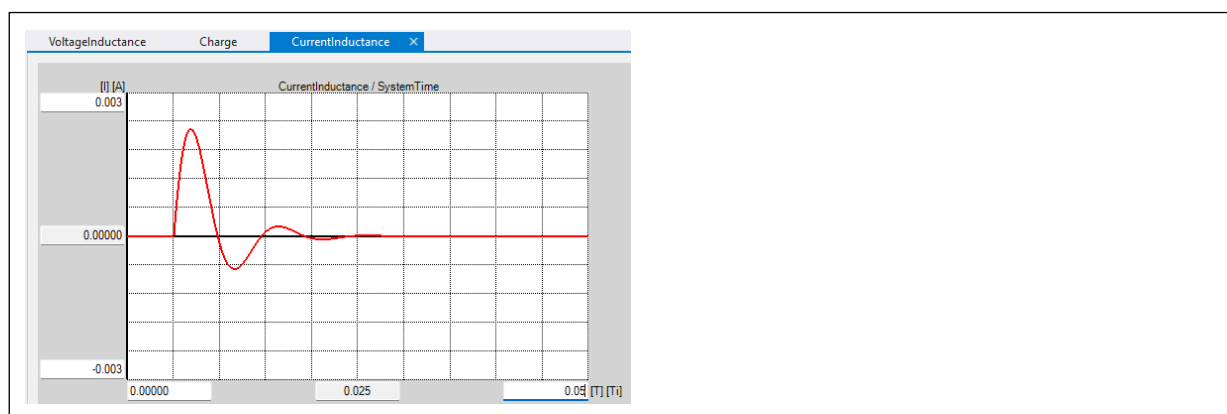


Figure 38 Current Inductance

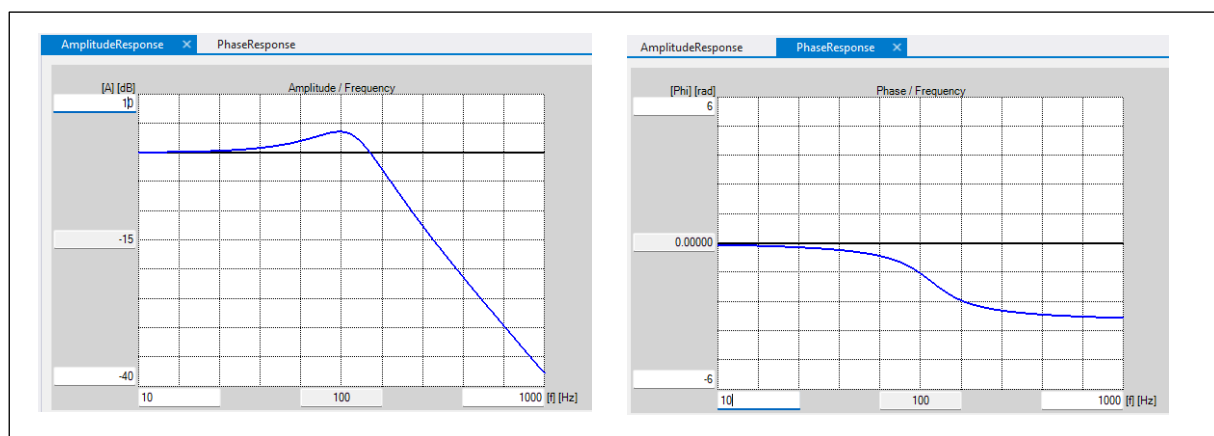


Figure 39 RLC Bode Diagram Curves

6.1.4 BraceCalculation Extended Implementation

This implementation requires the Professional version of **BraceCalculation**.

According to the laws of knots and meshes we draw following schematic:

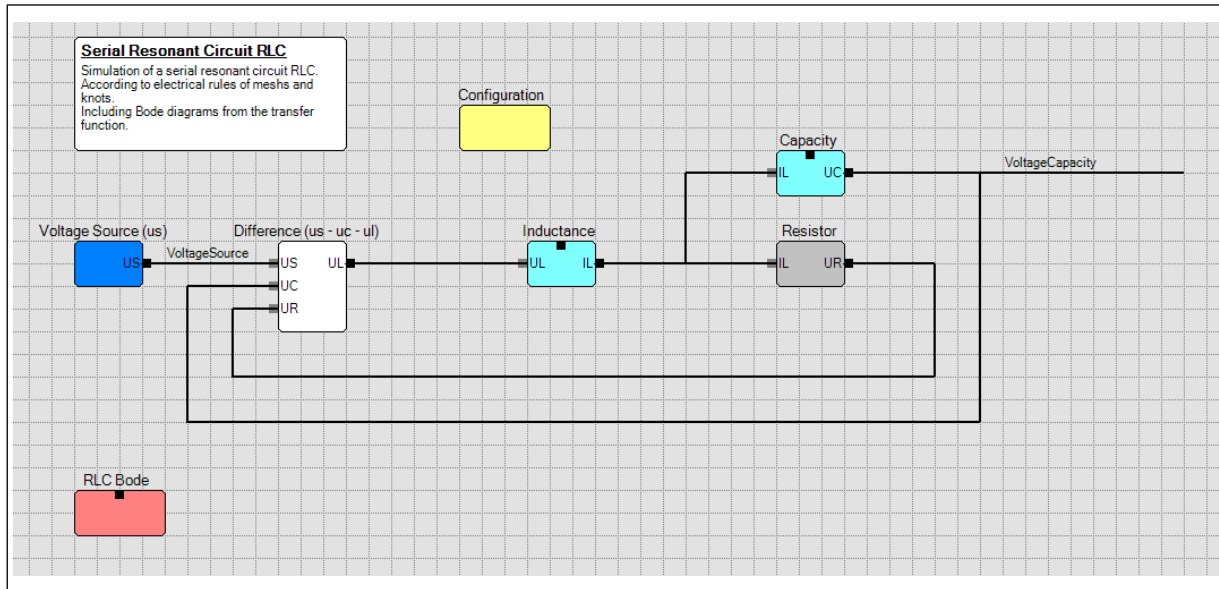


Figure 40 Schematic for a serial oscillation circuit

Timing

- One Tick Duration: 5E-8 second
- Sequence Duration: 1000000 ticks
- Number of Sequences: 1

Elements

Name	Pins	Curves	Tracks	Remarks
VoltageSource	Output US	VoltageSource (SystemTime)	Read RLC_Parameters	Voltage switches on after 10% simulation time.
Difference	Input US Input UC Input UR Output UI	VoltageSource (SystemTime) VoltageCapacity (SystemTime) VoltageResistor (SystemTime) VoltageInductance (SystemTime)		Voltage differences according to mesh.
Inductance	Input UL Output IL	VoltageInductance (SystemTime) CurrentInductance (SystemTime)	Read RLC_Parameters	Current is an integral from the voltage.
Capacity	Input IL Output UC	CurrentInductance (SystemTime) VoltageCapacity (SystemTime)	Read RLC_Parameters	Voltage is an integral from the current.
Resistor	Input IL Output UR	CurrentInductance (SystemTime) VoltageResistor (SystemTime)	Read RLC_Parameters	Voltage is linear from the current.
RLC Bode	Internal AR Internal PR	AmplitudeResponse (User def, 1..1000Hz log) PhaseResponse (User def, 1..1000Hz log)	Read RLC_Parameters	Static calculated transfer function.
Configuration			Write RLC_Parameters	Definition of RLC parameters for all curves.



Configuration User Code

```
public void Init(Int64 maxTicks, double tickTime)
{
    //
    // Track parameter:
    // string  text    [max 256]
    // byte[]  byteArray [max 1024]
    // double[] dArray  [max 256]
    // Int64[] iArray   [max 256]
    // double  dValueA
    // double  dValueB
    // Int64   iValueA
    // Int64   iValueB
    //
    var trackParam = new SEngineTrack();
    trackParam.dArray = new double[15]; // must be allocated

    // Resistor values
    trackParam.dArray[0] = 500; // R0
    trackParam.dArray[1] = 600; // R1
    trackParam.dArray[2] = 700; // R2
    trackParam.dArray[3] = 800; // R3
    trackParam.dArray[4] = 900; // R4

    // Inductance values
    trackParam.dArray[5] = 2.0; // L0
    trackParam.dArray[6] = 2.5; // L1
    trackParam.dArray[7] = 3.0; // L2
    trackParam.dArray[8] = 3.5; // L3
    trackParam.dArray[9] = 4.0; // L4

    // Capacity values
    trackParam.dArray[10] = 1E-6; // C0
    trackParam.dArray[11] = 2E-6; // C1
    trackParam.dArray[12] = 3E-6; // C2
    trackParam.dArray[13] = 4E-6; // C3
    trackParam.dArray[14] = 5E-6; // C4

    //
    // Write Track parameter to other Element
    //
    EngineTrack.Write(100001, trackParam);
    EngineTrack.WriteLocalEvent("RLC parameters distributed.", 6, TrackMsgType.Information);
}

public void Loop(Int64 maxTicks, Int64 curTicks, double curTime, double tickTime)
{
}

public void Post(Int64 maxTicks, double tickTime)
{
}
```



Voltage Source (us) User Code

```
private double us = 0;

public void Init(Int64 maxTicks, double tickTime)
{
}

public void Loop(Int64 maxTicks, Int64 curTicks, double curTime, double tickTime)
{
    //
    // We create a step from 0 to 5.0V after 10% time
    //
    if(curTicks < (maxTicks / 10))
    {
        us = 0;
    }
    else if (curTicks == (maxTicks / 10))
    {
        EngineTrack.WriteGlobalEvent("Voltage source is switching on.", TrackMsgType.Information);
        EngineTrack.WriteLocalEvent("Voltage source switches on.", 1, TrackMsgType.Information);
        us = 2.5;
    }
    else
    {
        us = 5.0;
    }

    OUT_VoltageSource.SetYDouble(us);
}

public void Post(Int64 maxTicks, double tickTime)
{
}
```



Difference (us – uc – ul) User Code

```
public void Init(Int64 maxTicks, double tickTime)
{
}

public void Loop(Int64 maxTicks, Int64 curTicks, double curTime, double tickTime)
{
    //
    // We create the difference: ur = us - ul - uc
    //
    var us = IN_VoltageSource.GetYDouble();

    var ur0 = IN_VoltageResistor.GetYDouble(0);
    var ur1 = IN_VoltageResistor.GetYDouble(1);
    var ur2 = IN_VoltageResistor.GetYDouble(2);
    var ur3 = IN_VoltageResistor.GetYDouble(3);
    var ur4 = IN_VoltageResistor.GetYDouble(4);

    var uc0 = IN_VoltageCapacity.GetYDouble(0);
    var uc1 = IN_VoltageCapacity.GetYDouble(1);
    var uc2 = IN_VoltageCapacity.GetYDouble(2);
    var uc3 = IN_VoltageCapacity.GetYDouble(3);
    var uc4 = IN_VoltageCapacity.GetYDouble(4);

    OUT_VoltageInductance.SetYDouble(0, us - ur0 - uc0);
    OUT_VoltageInductance.SetYDouble(1, us - ur1 - uc1);
    OUT_VoltageInductance.SetYDouble(2, us - ur2 - uc2);
    OUT_VoltageInductance.SetYDouble(3, us - ur3 - uc3);
    OUT_VoltageInductance.SetYDouble(4, us - ur4 - uc4);
}

public void Post(Int64 maxTicks, double tickTime)
{
}
```



Inductance User Code

```
private double L0 = 1.0;
private double L1 = 1.0;
private double L2 = 1.0;
private double L3 = 1.0;
private double L4 = 1.0;
private int key0 = 0;
private int key1 = 0;
private int key2 = 0;
private int key3 = 0;
private int key4 = 0;

public void Init(Int64 maxTicks, double tickTime)
{
    //
    // Read Track parameter from Configuration Element
    //
    var trackParam = new SEngineTrack();
    EngineTrack.Read(100001, ref trackParam);
    L0 = trackParam.dArray[5];
    L1 = trackParam.dArray[6];
    L2 = trackParam.dArray[7];
    L3 = trackParam.dArray[8];
    L4 = trackParam.dArray[9];

    EngineTrack.WriteLocalEvent("Inductance Element got L parameters.", 4, TrackMsgType.Information);

    key0 = EngineLibrary.Operation.CreateIntegration(0);
    key1 = EngineLibrary.Operation.CreateIntegration(0);
    key2 = EngineLibrary.Operation.CreateIntegration(0);
    key3 = EngineLibrary.Operation.CreateIntegration(0);
    key4 = EngineLibrary.Operation.CreateIntegration(0);
}

public void Loop(Int64 maxTicks, Int64 curTicks, double curTime, double tickTime)
{
    //
    // We create the inductance current
    //
    EngineLibrary.Operation.ActIntegrate(key0, IN_VoltageInductance, 0, INTERNAL_Charge, 0);
    EngineLibrary.Operation.ActIntegrate(key1, IN_VoltageInductance, 1, INTERNAL_Charge, 1);
    EngineLibrary.Operation.ActIntegrate(key2, IN_VoltageInductance, 2, INTERNAL_Charge, 2);
    EngineLibrary.Operation.ActIntegrate(key3, IN_VoltageInductance, 3, INTERNAL_Charge, 3);
    EngineLibrary.Operation.ActIntegrate(key4, IN_VoltageInductance, 4, INTERNAL_Charge, 4);

    var volt0 = INTERNAL_Charge.GetYDouble(0);
    var volt1 = INTERNAL_Charge.GetYDouble(1);
    var volt2 = INTERNAL_Charge.GetYDouble(2);
    var volt3 = INTERNAL_Charge.GetYDouble(3);
    var volt4 = INTERNAL_Charge.GetYDouble(4);

    OUT_CurrentInductance.SetYDouble(0, volt0 / L0);
    OUT_CurrentInductance.SetYDouble(1, volt1 / L1);
    OUT_CurrentInductance.SetYDouble(2, volt2 / L2);
    OUT_CurrentInductance.SetYDouble(3, volt3 / L3);
    OUT_CurrentInductance.SetYDouble(4, volt4 / L4);
}

public void Post(Int64 maxTicks, double tickTime)
{
}
```



Capacity User Code

```
private double C0 = 1E-6;
private double C1 = 1E-6;
private double C2 = 1E-6;
private double C3 = 1E-6;
private double C4 = 1E-6;
private int key0 = 0;
private int key1 = 0;
private int key2 = 0;
private int key3 = 0;
private int key4 = 0;

public void Init(Int64 maxTicks, double tickTime)
{
    //
    // Read Track parameter from Configuration Element
    //
    var trackParam = new SEngineTrack();
    EngineTrack.Read(100001, ref trackParam);
    C0 = trackParam.dArray[10];
    C1 = trackParam.dArray[11];
    C2 = trackParam.dArray[12];
    C3 = trackParam.dArray[13];
    C4 = trackParam.dArray[14];

    EngineTrack.WriteLocalEvent("Capacity Element got C parameters.", 5, TrackMsgType.Information);

    key0 = EngineLibrary.Operation.CreateIntegration(0);
    key1 = EngineLibrary.Operation.CreateIntegration(0);
    key2 = EngineLibrary.Operation.CreateIntegration(0);
    key3 = EngineLibrary.Operation.CreateIntegration(0);
    key4 = EngineLibrary.Operation.CreateIntegration(0);
}

public void Loop(Int64 maxTicks, Int64 curTicks, double curTime, double tickTime)
{
    // Coulomb to internal curve
    var current0 = IN_CurrentInductance.GetYDouble(0);
    var current1 = IN_CurrentInductance.GetYDouble(1);
    var current2 = IN_CurrentInductance.GetYDouble(2);
    var current3 = IN_CurrentInductance.GetYDouble(3);
    var current4 = IN_CurrentInductance.GetYDouble(4);

    INTERNAL_Coulomb.SetYDouble(0, current0 / C0);
    INTERNAL_Coulomb.SetYDouble(1, current1 / C1);
    INTERNAL_Coulomb.SetYDouble(2, current2 / C2);
    INTERNAL_Coulomb.SetYDouble(3, current3 / C3);
    INTERNAL_Coulomb.SetYDouble(4, current4 / C4);

    // Integrate to output curve
    EngineLibrary.Operation.ActIntegrate(key0, INTERNAL_Coulomb, 0, OUT_VoltageCapacity, 0);
    EngineLibrary.Operation.ActIntegrate(key1, INTERNAL_Coulomb, 1, OUT_VoltageCapacity, 1);
    EngineLibrary.Operation.ActIntegrate(key2, INTERNAL_Coulomb, 2, OUT_VoltageCapacity, 2);
    EngineLibrary.Operation.ActIntegrate(key3, INTERNAL_Coulomb, 3, OUT_VoltageCapacity, 3);
    EngineLibrary.Operation.ActIntegrate(key4, INTERNAL_Coulomb, 4, OUT_VoltageCapacity, 4);
}

public void Post(Int64 maxTicks, double tickTime)
{
}
```



Resistor User Code

```
private double R0 = 1000.0;
private double R1 = 500.0;
private double R2 = 500.0;
private double R3 = 500.0;
private double R4 = 500.0;

public void Init(Int64 maxTicks, double tickTime)
{
    var trackParam = new SEngineTrack();

    //
    // Read Track parameter from Configuration Element
    //
    EngineTrack.Read(100001, ref trackParam);
    R0 = trackParam.dArray[0];
    R1 = trackParam.dArray[1];
    R2 = trackParam.dArray[2];
    R3 = trackParam.dArray[3];
    R4 = trackParam.dArray[4];

    EngineTrack.WriteLocalEvent("Resistor Element got R parameters.", 3, TrackMsgType.Information);
}

public void Loop(Int64 maxTicks, Int64 curTicks, double curTime, double tickTime)
{
    //
    // We create the resistor currents
    //
    var cur0 = IN_CurrentInductance.GetYDouble(0);
    var cur1 = IN_CurrentInductance.GetYDouble(1);
    var cur2 = IN_CurrentInductance.GetYDouble(2);
    var cur3 = IN_CurrentInductance.GetYDouble(3);
    var cur4 = IN_CurrentInductance.GetYDouble(4);

    OUT_VoltageResistor.SetYDouble(0, cur0 * R0);
    OUT_VoltageResistor.SetYDouble(1, cur1 * R1);
    OUT_VoltageResistor.SetYDouble(2, cur2 * R2);
    OUT_VoltageResistor.SetYDouble(3, cur3 * R3);
    OUT_VoltageResistor.SetYDouble(4, cur4 * R4);
}

public void Post(Int64 maxTicks, double tickTime)
{
}
```



RLC Bode User Code

```
public void Init(Int64 maxTicks, double tickTime)
{
    var trackParam = new SEngineTrack();
    // Read Track parameter from Configuration Element
    EngineTrack.Read(100001, ref trackParam);
    var R0 = trackParam.dArray[0];
    var R1 = trackParam.dArray[1];
    var R2 = trackParam.dArray[2];
    var R3 = trackParam.dArray[3];
    var R4 = trackParam.dArray[4];
    var L0 = trackParam.dArray[5];
    var L1 = trackParam.dArray[6];
    var L2 = trackParam.dArray[7];
    var L3 = trackParam.dArray[8];
    var L4 = trackParam.dArray[9];
    var C0 = trackParam.dArray[10];
    var C1 = trackParam.dArray[11];
    var C2 = trackParam.dArray[12];
    var C3 = trackParam.dArray[13];
    var C4 = trackParam.dArray[14];

    EngineTrack.WriteLocalEvent("Bode Element got RLC parameters.", 7, TrackMsgType.Information);

    // Set the laplace parameters
    var coA = new double[] {1.0};
    var coB0 = new double[] {1.0, R0 * C0, L0 * C0};
    var coB1 = new double[] {1.0, R1 * C1, L1 * C1};
    var coB2 = new double[] {1.0, R2 * C2, L2 * C2};
    var coB3 = new double[] {1.0, R3 * C3, L3 * C3};
    var coB4 = new double[] {1.0, R4 * C4, L4 * C4};
    // Create the polynoms
    var rlcBode0 = EngineLibrary.Filter.CreateAnalogPolynom(coA, coB0);
    var rlcBode1 = EngineLibrary.Filter.CreateAnalogPolynom(coA, coB1);
    var rlcBode2 = EngineLibrary.Filter.CreateAnalogPolynom(coA, coB2);
    var rlcBode3 = EngineLibrary.Filter.CreateAnalogPolynom(coA, coB3);
    var rlcBode4 = EngineLibrary.Filter.CreateAnalogPolynom(coA, coB4);

    // Create Amplitude response curves in Dezibel
    EngineLibrary.Filter.CurveFilterAmplitudeResponse(rlcBode0, INTERNAL_AmplitudeResponse, 0);
    EngineLibrary.Operation.CurveConvertDB(INTERNAL_AmplitudeResponse, 0, INTERNAL_AmplitudeResponse, 0);
    EngineLibrary.Filter.CurveFilterAmplitudeResponse(rlcBode1, INTERNAL_AmplitudeResponse, 1);
    EngineLibrary.Operation.CurveConvertDB(INTERNAL_AmplitudeResponse, 1, INTERNAL_AmplitudeResponse, 1);
    EngineLibrary.Filter.CurveFilterAmplitudeResponse(rlcBode2, INTERNAL_AmplitudeResponse, 2);
    EngineLibrary.Operation.CurveConvertDB(INTERNAL_AmplitudeResponse, 2, INTERNAL_AmplitudeResponse, 2);
    EngineLibrary.Filter.CurveFilterAmplitudeResponse(rlcBode3, INTERNAL_AmplitudeResponse, 3);
    EngineLibrary.Operation.CurveConvertDB(INTERNAL_AmplitudeResponse, 3, INTERNAL_AmplitudeResponse, 3);
    EngineLibrary.Filter.CurveFilterAmplitudeResponse(rlcBode4, INTERNAL_AmplitudeResponse, 4);
    EngineLibrary.Operation.CurveConvertDB(INTERNAL_AmplitudeResponse, 4, INTERNAL_AmplitudeResponse, 4);

    // Create Phase response curves
    EngineLibrary.Filter.CurveFilterPhaseResponse(rlcBode0, INTERNAL_PhaseResponse, 0);
    EngineLibrary.Filter.CurveFilterPhaseResponse(rlcBode1, INTERNAL_PhaseResponse, 1);
    EngineLibrary.Filter.CurveFilterPhaseResponse(rlcBode2, INTERNAL_PhaseResponse, 2);
    EngineLibrary.Filter.CurveFilterPhaseResponse(rlcBode3, INTERNAL_PhaseResponse, 3);
    EngineLibrary.Filter.CurveFilterPhaseResponse(rlcBode4, INTERNAL_PhaseResponse, 4);
}

public void Loop(Int64 maxTicks, Int64 curTicks, double curTime, double tickTime)
{
}

public void Post(Int64 maxTicks, double tickTime)
{
}
```

6.1.5 BraceCalculation Extended Simulation

After Build and Start the following curves are visible in the different scopes.



Figure 41 Voltage Source

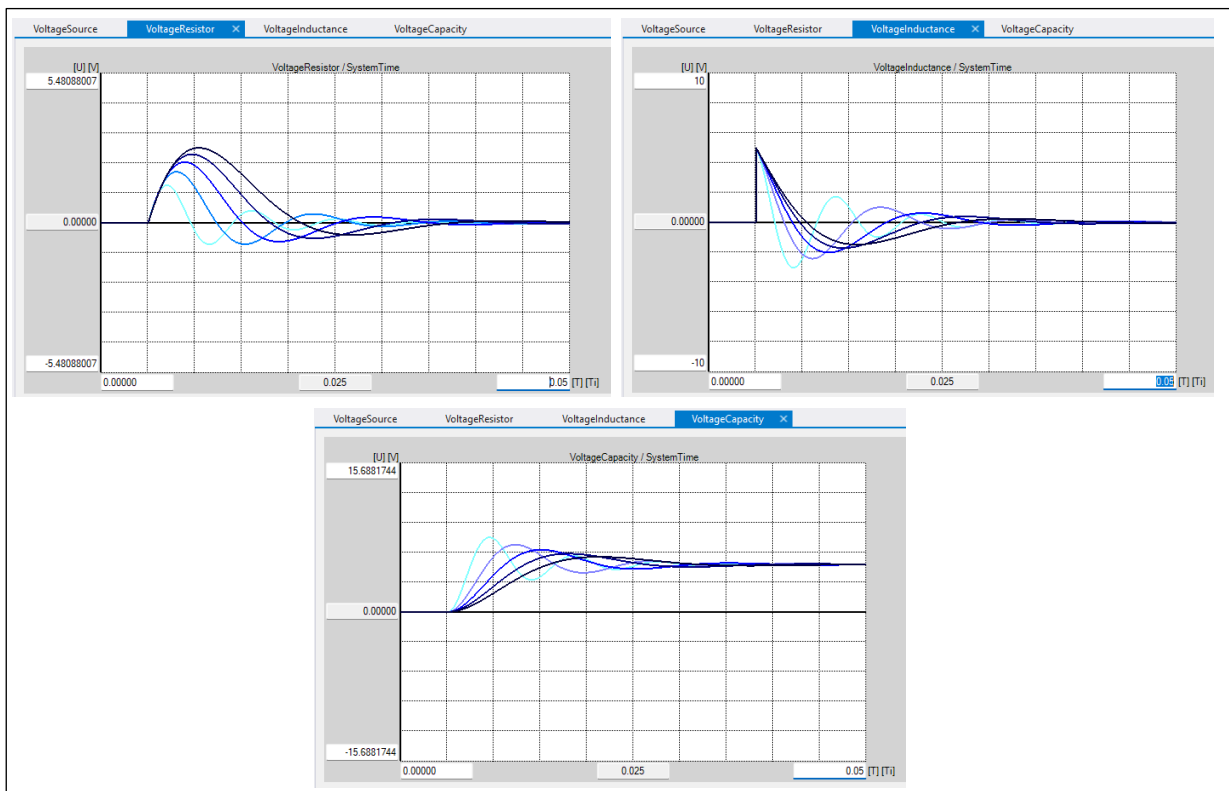


Figure 42 Voltages RLC

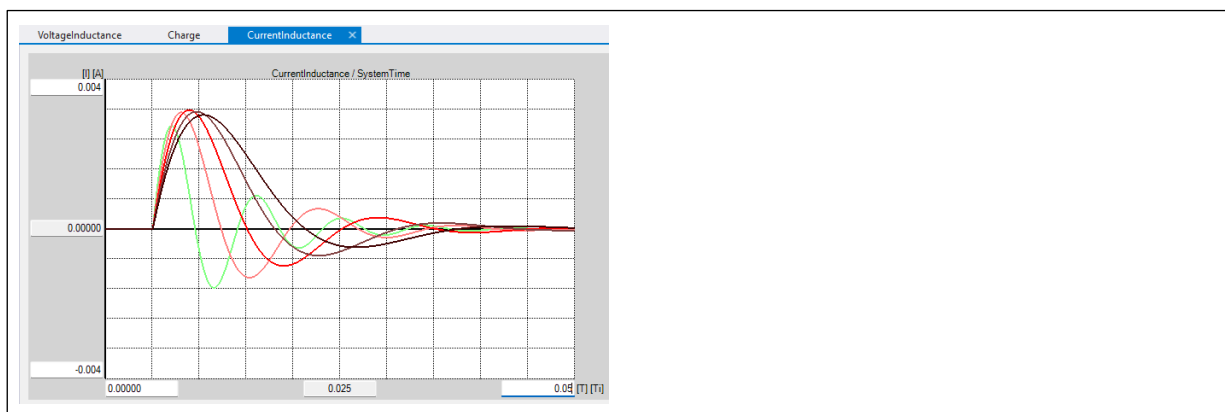


Figure 43 Current Inductance

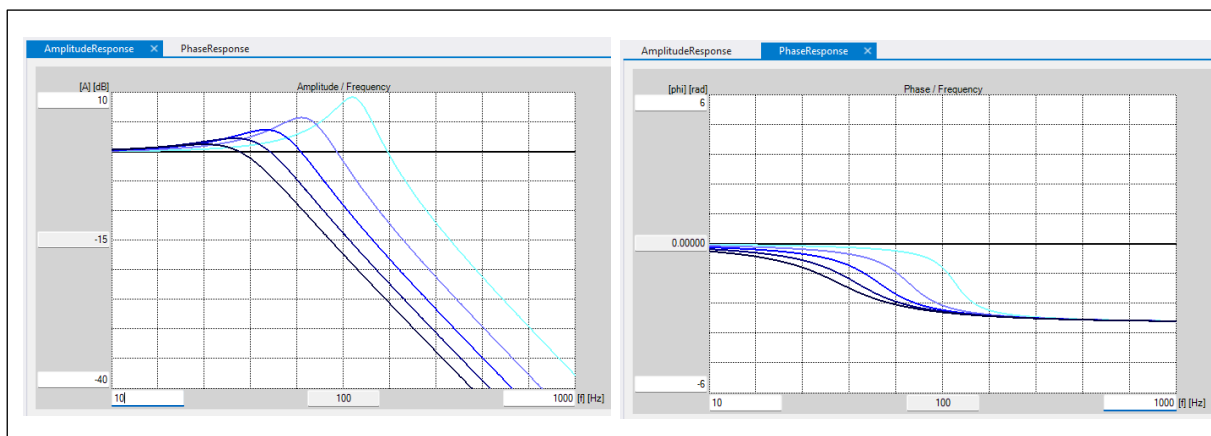


Figure 44 RLC Bode Diagram Curves



6.1.6 BraceCalculation Assembly Implementation

This implementation with an assembly requires the Professional version of **BraceCalculation**.

According to the laws of knots and meshes we draw following schematic:

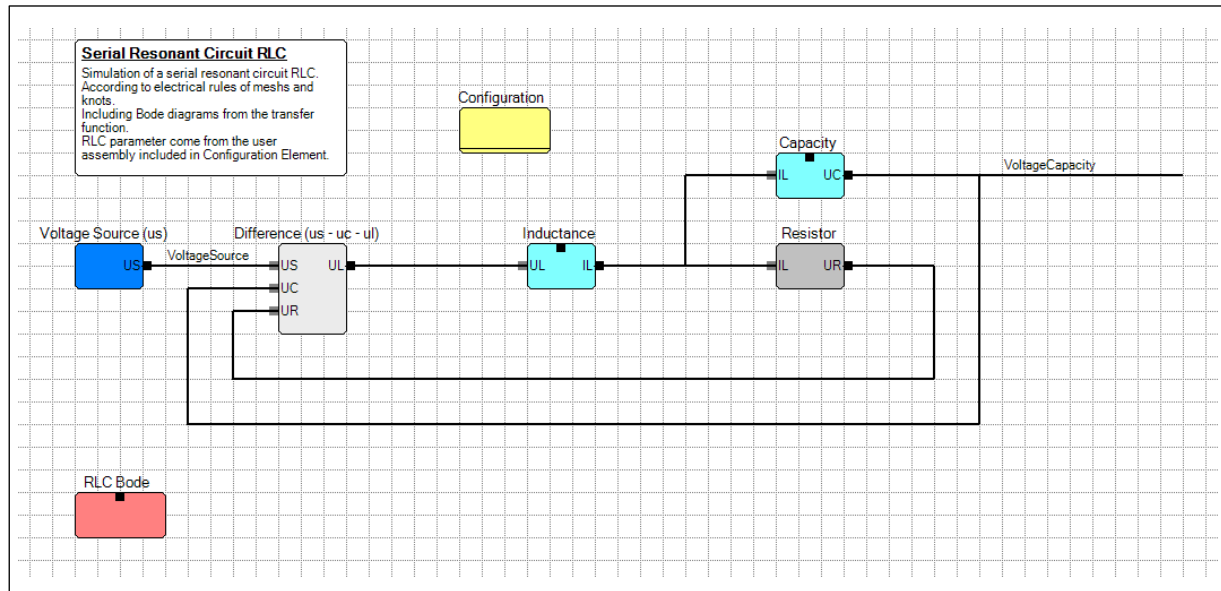


Figure 45 Schematic for a serial oscillation circuit

Timing

- One Tick Duration: 5E-8 second
- Sequence Duration: 2000000 ticks
- Number of Sequences: 1

Elements

Name	Pins	Curves	Tracks	Remarks
VoltageSource	Output US	VoltageSource (SystemTime)	Read RLC_Parameters	Voltage switches on after 10% simulation time.
Difference	Input US Input UC Input UR Output UI	VoltageSource (SystemTime) VoltageCapacity (SystemTime) VoltageResistor (SystemTime) VoltageInductance (SystemTime)		Voltage differences according to mesh.
Inductance	Input UL Output IL	VoltageInductance (SystemTime) CurrentInductance (SystemTime)	Read RLC_Parameters	Current is an integral from the voltage.
Capacity	Input IL Output UC	CurrentInductance (SystemTime) VoltageCapacity (SystemTime)	Read RLC_Parameters	Voltage is an integral from the current.
Resistor	Input IL Output UR	CurrentInductance (SystemTime) VoltageResistor (SystemTime)	Read RLC_Parameters	Voltage is linear from the current.
RLC Bode	Internal AR Internal PR	AmplitudeResponse (User def, 1..1000Hz log) PhaseResponse (User def, 1..1000Hz log)	Read RLC_Parameters	Static calculated transfer function.
Configuration			Write RLC_Parameters	Definition of RLC parameters for all curves from a custom .NET Framework DLL.



Configuration User Code

```
public void Init(Int64 maxTicks, double tickTime)
{
}

public void Loop(Int64 maxTicks, Int64 curTicks, double curTime, double tickTime)
{
}

public void Post(Int64 maxTicks, double tickTime)
{
}
```

Remarks

- Configuration Element doesn't need any code.
- All the providing parameters come from the Assembly.



Voltage Source (us) User Code Init()

```
private double us = 0;
private double Volt = 0;
private double Freq0 = 0;
private double Freq1 = 0;
private int Ticks0 = 0;
private int Ticks1 = 0;
private int Ticks2 = 0;
private int Ticks3 = 0;

public void Init(Int64 maxTicks, double tickTime)
{
    //
    // Receive Track parameter from Assembly
    //
    var assemblyParam = new SEngineTrack();
    EngineTrack.Receive(100001, ref assemblyParam);

    Volt = assemblyParam.dArray[15];
    Freq0 = (int)assemblyParam.dArray[16];
    Freq1 = (int)assemblyParam.dArray[17];

    Ticks0 = (int)assemblyParam.dArray[18];
    Ticks1 = (int)assemblyParam.dArray[19];
    Ticks2 = (int)assemblyParam.dArray[20];
    Ticks3 = (int)assemblyParam.dArray[21];

    EngineTrack.WriteLocalEvent("Voltage Source Element got parameters.", 1, TrackMsgType.Information);
}
```

Remarks

- The voltage and frequency parameters come from the Assembly via Receive().



Voltage Source (us) User Code Loop() and Post()

```
public void Loop(Int64 maxTicks, Int64 curTicks, double curTime, double tickTime)
{
    //
    // We create a step from 0 to Volt after first percent of time
    //
    if(curTicks < Ticks0)
    {
        us = 0;
    }
    else if (curTicks == Ticks0)
    {
        EngineTrack.WriteGlobalEvent("Voltage source is switching on.", TrackMsgType.Information);
        EngineTrack.WriteLocalEvent("Voltage source switches on.", 1, TrackMsgType.Information);
    }
    else if (curTicks < Ticks1)
    {
        us = Volt;
    }
    else if (curTicks == Ticks1)
    {
        EngineTrack.WriteLocalEvent("Voltage source switches off.", 1, TrackMsgType.Information);
    }
    else if (curTicks < Ticks2)
    {
        us = 0;
    }
    else if (curTicks == Ticks2)
    {
        EngineTrack.WriteLocalEvent("Voltage source witch sinus1.", 1, TrackMsgType.Information);
    }
    else if (curTicks < Ticks3)
    {
        us = Volt * Math.Sin(Math.PI * 2 * Freq0 * curTime);
    }
    else if (curTicks == Ticks3)
    {
        EngineTrack.WriteLocalEvent("Voltage source with sinus2.", 1, TrackMsgType.Information);
    }
    else if (curTicks < maxTicks)
    {
        us = Volt * Math.Sin(Math.PI * 2 * Freq1 * curTime);
    }
    else
    {
        us = Volt;
    }

    OUT_VoltageSource.SetYDouble(us);
}

public void Post(Int64 maxTicks, double tickTime)
{
}
```

Remarks

- We set different source outputs according to the received time ticks from the assembly.



Difference (us – uc – ul) User Code

```
public void Init(Int64 maxTicks, double tickTime)
{
}

public void Loop(Int64 maxTicks, Int64 curTicks, double curTime, double tickTime)
{
    //
    // We create the difference: ur = us - ul - uc
    //
    var us = IN_VoltageSource.GetYDouble();

    var ur0 = IN_VoltageResistor.GetYDouble(0);
    var ur1 = IN_VoltageResistor.GetYDouble(1);
    var ur2 = IN_VoltageResistor.GetYDouble(2);
    var ur3 = IN_VoltageResistor.GetYDouble(3);
    var ur4 = IN_VoltageResistor.GetYDouble(4);

    var uc0 = IN_VoltageCapacity.GetYDouble(0);
    var uc1 = IN_VoltageCapacity.GetYDouble(1);
    var uc2 = IN_VoltageCapacity.GetYDouble(2);
    var uc3 = IN_VoltageCapacity.GetYDouble(3);
    var uc4 = IN_VoltageCapacity.GetYDouble(4);

    OUT_VoltageInductance.SetYDouble(0, us - ur0 - uc0);
    OUT_VoltageInductance.SetYDouble(1, us - ur1 - uc1);
    OUT_VoltageInductance.SetYDouble(2, us - ur2 - uc2);
    OUT_VoltageInductance.SetYDouble(3, us - ur3 - uc3);
    OUT_VoltageInductance.SetYDouble(4, us - ur4 - uc4);
}

public void Post(Int64 maxTicks, double tickTime)
{
}
```



Inductance User Code

```
private double L0 = 1.0;
private double L1 = 1.0;
private double L2 = 1.0;
private double L3 = 1.0;
private double L4 = 1.0;
private int key0 = 0;
private int key1 = 0;
private int key2 = 0;
private int key3 = 0;
private int key4 = 0;

public void Init(Int64 maxTicks, double tickTime)
{
    //
    // Read Track parameter from Configuration Element
    //
    var trackParam = new SEngineTrack();
    EngineTrack.Receive(100001, ref trackParam);
    L0 = trackParam.dArray[5];
    L1 = trackParam.dArray[6];
    L2 = trackParam.dArray[7];
    L3 = trackParam.dArray[8];
    L4 = trackParam.dArray[9];

    EngineTrack.WriteLocalEvent("Inductance Element got L parameters.", 4, TrackMsgType.Information);

    key0 = EngineLibrary.Operation.CreateIntegration(0);
    key1 = EngineLibrary.Operation.CreateIntegration(0);
    key2 = EngineLibrary.Operation.CreateIntegration(0);
    key3 = EngineLibrary.Operation.CreateIntegration(0);
    key4 = EngineLibrary.Operation.CreateIntegration(0);
}

public void Loop(Int64 maxTicks, Int64 curTicks, double curTime, double tickTime)
{
    //
    // We create the inductance current
    //
    EngineLibrary.Operation.ActIntegrate(key0, IN_VoltageInductance, 0, INTERNAL_Charge, 0);
    EngineLibrary.Operation.ActIntegrate(key1, IN_VoltageInductance, 1, INTERNAL_Charge, 1);
    EngineLibrary.Operation.ActIntegrate(key2, IN_VoltageInductance, 2, INTERNAL_Charge, 2);
    EngineLibrary.Operation.ActIntegrate(key3, IN_VoltageInductance, 3, INTERNAL_Charge, 3);
    EngineLibrary.Operation.ActIntegrate(key4, IN_VoltageInductance, 4, INTERNAL_Charge, 4);

    var volt0 = INTERNAL_Charge.GetYDouble(0);
    var volt1 = INTERNAL_Charge.GetYDouble(1);
    var volt2 = INTERNAL_Charge.GetYDouble(2);
    var volt3 = INTERNAL_Charge.GetYDouble(3);
    var volt4 = INTERNAL_Charge.GetYDouble(4);

    OUT_CurrentInductance.SetYDouble(0, volt0 / L0);
    OUT_CurrentInductance.SetYDouble(1, volt1 / L1);
    OUT_CurrentInductance.SetYDouble(2, volt2 / L2);
    OUT_CurrentInductance.SetYDouble(3, volt3 / L3);
    OUT_CurrentInductance.SetYDouble(4, volt4 / L4);
}

public void Post(Int64 maxTicks, double tickTime)
{
}
```



Capacity User Code

```
private double C0 = 1E-6;
private double C1 = 1E-6;
private double C2 = 1E-6;
private double C3 = 1E-6;
private double C4 = 1E-6;
private int key0 = 0;
private int key1 = 0;
private int key2 = 0;
private int key3 = 0;
private int key4 = 0;

public void Init(Int64 maxTicks, double tickTime)
{
    //
    // Read Track parameter from Configuration Element
    //
    var trackParam = new SEngineTrack();
    EngineTrack.Receive(100001, ref trackParam);
    C0 = trackParam.dArray[10];
    C1 = trackParam.dArray[11];
    C2 = trackParam.dArray[12];
    C3 = trackParam.dArray[13];
    C4 = trackParam.dArray[14];

    EngineTrack.WriteLocalEvent("Capacity Element got C parameters.", 5, TrackMsgType.Information);

    key0 = EngineLibrary.Operation.CreateIntegration(0);
    key1 = EngineLibrary.Operation.CreateIntegration(0);
    key2 = EngineLibrary.Operation.CreateIntegration(0);
    key3 = EngineLibrary.Operation.CreateIntegration(0);
    key4 = EngineLibrary.Operation.CreateIntegration(0);
}

public void Loop(Int64 maxTicks, Int64 curTicks, double curTime, double tickTime)
{
    // Coulomb to internal curve
    var current0 = IN_CurrentInductance.GetYDouble(0);
    var current1 = IN_CurrentInductance.GetYDouble(1);
    var current2 = IN_CurrentInductance.GetYDouble(2);
    var current3 = IN_CurrentInductance.GetYDouble(3);
    var current4 = IN_CurrentInductance.GetYDouble(4);

    INTERNAL_Coulomb.SetYDouble(0, current0 / C0);
    INTERNAL_Coulomb.SetYDouble(1, current1 / C1);
    INTERNAL_Coulomb.SetYDouble(2, current2 / C2);
    INTERNAL_Coulomb.SetYDouble(3, current3 / C3);
    INTERNAL_Coulomb.SetYDouble(4, current4 / C4);

    // Integrate to output curve
    EngineLibrary.Operation.ActIntegrate(key0, INTERNAL_Coulomb, 0, OUT_VoltageCapacity, 0);
    EngineLibrary.Operation.ActIntegrate(key1, INTERNAL_Coulomb, 1, OUT_VoltageCapacity, 1);
    EngineLibrary.Operation.ActIntegrate(key2, INTERNAL_Coulomb, 2, OUT_VoltageCapacity, 2);
    EngineLibrary.Operation.ActIntegrate(key3, INTERNAL_Coulomb, 3, OUT_VoltageCapacity, 3);
    EngineLibrary.Operation.ActIntegrate(key4, INTERNAL_Coulomb, 4, OUT_VoltageCapacity, 4);
}

public void Post(Int64 maxTicks, double tickTime)
{
}
```



Resistor User Code

```
private double R0 = 1000.0;
private double R1 = 500.0;
private double R2 = 500.0;
private double R3 = 500.0;
private double R4 = 500.0;

public void Init(Int64 maxTicks, double tickTime)
{
    var trackParam = new SEngineTrack();

    //
    // Read Track parameter from Configuration Element
    //
    EngineTrack.Read(100001, ref trackParam);
    R0 = trackParam.dArray[0];
    R1 = trackParam.dArray[1];
    R2 = trackParam.dArray[2];
    R3 = trackParam.dArray[3];
    R4 = trackParam.dArray[4];

    EngineTrack.WriteLocalEvent("Resistor Element got R parameters.", 3, TrackMsgType.Information);
}

public void Loop(Int64 maxTicks, Int64 curTicks, double curTime, double tickTime)
{
    //
    // We create the resistor currents
    //
    var cur0 = IN_CurrentInductance.GetYDouble(0);
    var cur1 = IN_CurrentInductance.GetYDouble(1);
    var cur2 = IN_CurrentInductance.GetYDouble(2);
    var cur3 = IN_CurrentInductance.GetYDouble(3);
    var cur4 = IN_CurrentInductance.GetYDouble(4);

    OUT_VoltageResistor.SetYDouble(0, cur0 * R0);
    OUT_VoltageResistor.SetYDouble(1, cur1 * R1);
    OUT_VoltageResistor.SetYDouble(2, cur2 * R2);
    OUT_VoltageResistor.SetYDouble(3, cur3 * R3);
    OUT_VoltageResistor.SetYDouble(4, cur4 * R4);
}

public void Post(Int64 maxTicks, double tickTime)
{
}
```



RLC Bode User Code

```
public void Init(Int64 maxTicks, double tickTime)
{
    // Read Track parameter from Configuration Element
    var trackParam = new SEngineTrack();
    EngineTrack.Receive(100001, ref trackParam);
    var R0 = trackParam.dArray[0];
    var R1 = trackParam.dArray[1];
    var R2 = trackParam.dArray[2];
    var R3 = trackParam.dArray[3];
    var R4 = trackParam.dArray[4];
    var L0 = trackParam.dArray[5];
    var L1 = trackParam.dArray[6];
    var L2 = trackParam.dArray[7];
    var L3 = trackParam.dArray[8];
    var L4 = trackParam.dArray[9];
    var C0 = trackParam.dArray[10];
    var C1 = trackParam.dArray[11];
    var C2 = trackParam.dArray[12];
    var C3 = trackParam.dArray[13];
    var C4 = trackParam.dArray[14];

    EngineTrack.WriteLocalEvent("Bode Element got RLC parameters.", 7, TrackMsgType.Information);

    // Set the laplace parameters
    var coA = new double[] {1.0};
    var coB0 = new double[] {1.0, R0 * C0, L0 * C0};
    var coB1 = new double[] {1.0, R1 * C1, L1 * C1};
    var coB2 = new double[] {1.0, R2 * C2, L2 * C2};
    var coB3 = new double[] {1.0, R3 * C3, L3 * C3};
    var coB4 = new double[] {1.0, R4 * C4, L4 * C4};
    // Create the polynoms
    var rlcBode0 = EngineLibrary.Filter.CreateAnalogPolynom(coA, coB0);
    var rlcBode1 = EngineLibrary.Filter.CreateAnalogPolynom(coA, coB1);
    var rlcBode2 = EngineLibrary.Filter.CreateAnalogPolynom(coA, coB2);
    var rlcBode3 = EngineLibrary.Filter.CreateAnalogPolynom(coA, coB3);
    var rlcBode4 = EngineLibrary.Filter.CreateAnalogPolynom(coA, coB4);

    // Create Amplitude response curves in Dezibel
    EngineLibrary.Filter.CurveFilterAmplitudeResponse(rlcBode0, INTERNAL_AmplitudeResponse, 0);
    EngineLibrary.Operation.CurveConvertDB(INTERNAL_AmplitudeResponse, 0, INTERNAL_AmplitudeResponse, 0);
    EngineLibrary.Filter.CurveFilterAmplitudeResponse(rlcBode1, INTERNAL_AmplitudeResponse, 1);
    EngineLibrary.Operation.CurveConvertDB(INTERNAL_AmplitudeResponse, 1, INTERNAL_AmplitudeResponse, 1);
    EngineLibrary.Filter.CurveFilterAmplitudeResponse(rlcBode2, INTERNAL_AmplitudeResponse, 2);
    EngineLibrary.Operation.CurveConvertDB(INTERNAL_AmplitudeResponse, 2, INTERNAL_AmplitudeResponse, 2);
    EngineLibrary.Filter.CurveFilterAmplitudeResponse(rlcBode3, INTERNAL_AmplitudeResponse, 3);
    EngineLibrary.Operation.CurveConvertDB(INTERNAL_AmplitudeResponse, 3, INTERNAL_AmplitudeResponse, 3);
    EngineLibrary.Filter.CurveFilterAmplitudeResponse(rlcBode4, INTERNAL_AmplitudeResponse, 4);
    EngineLibrary.Operation.CurveConvertDB(INTERNAL_AmplitudeResponse, 4, INTERNAL_AmplitudeResponse, 4);

    // Create Phase response curves
    EngineLibrary.Filter.CurveFilterPhaseResponse(rlcBode0, INTERNAL_PhaseResponse, 0);
    EngineLibrary.Filter.CurveFilterPhaseResponse(rlcBode1, INTERNAL_PhaseResponse, 1);
    EngineLibrary.Filter.CurveFilterPhaseResponse(rlcBode2, INTERNAL_PhaseResponse, 2);
    EngineLibrary.Filter.CurveFilterPhaseResponse(rlcBode3, INTERNAL_PhaseResponse, 3);
    EngineLibrary.Filter.CurveFilterPhaseResponse(rlcBode4, INTERNAL_PhaseResponse, 4);
}

public void Loop(Int64 maxTicks, Int64 curTicks, double curTime, double tickTime)
{
}

public void Post(Int64 maxTicks, double tickTime)
{
}
```

6.1.7 BraceCalculation Assembly DLL Creation

To create an Assembly DLL for **BraceCalculation**, you need a VisualStudio installation which supports .NET Framework 4.8.

VisualStudio 2026 Assembly Project

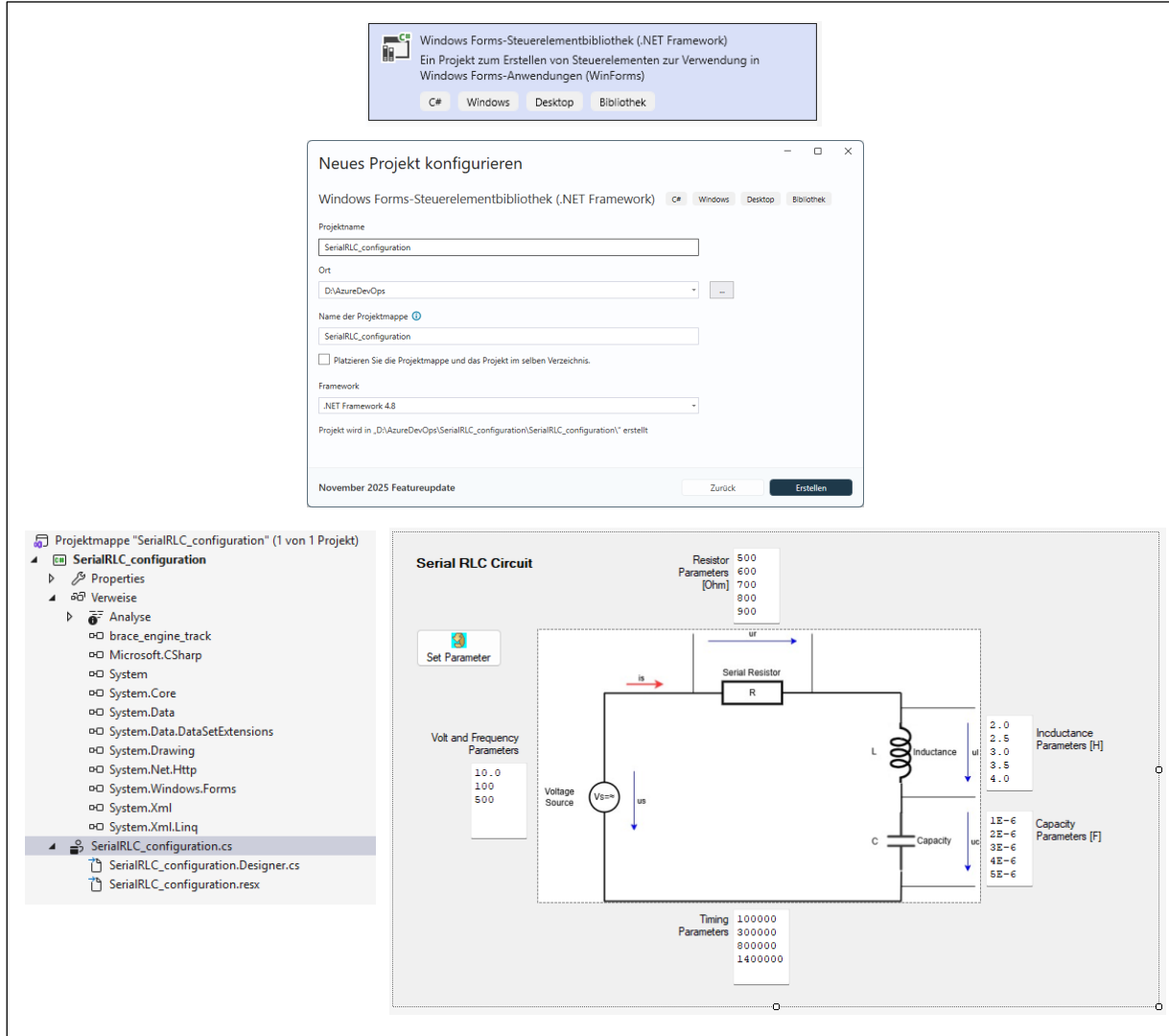


Figure 46 VisualStudio 2026 .NET Framework DLL Assembly SerialRLC_configuration

Remarks

- We create a C# .NET Framework DLL project.
- The GUI base element is a Window.Forms.UserControl.
- Following controls are included in the UserControl
 - Label for a title.
 - Image from the circuit we simulate.
 - Button to set new and changed parameters.
 - Textbox for source voltage and frequencies.
 - Textbox for timing ticks.
 - Textbox for resistor parameters.
 - Textbox for inductance parameters.
 - Textbox for capacitor parameters.



SerialRLC_configuration.cs code part 1:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using brace_engine_track;

//=====
namespace SerialRLC_configuration
//=====
{
    //=====
    public partial class SerialRLC_configuration: UserControl, IEngineTrackSlave
    //=====
    {
        private IEngineTrackInfo m_engineTrack;
        private int m_keyElement = 0;
        private List<double> m_parameters = new List<double>();

        //-----
        public SerialRLC_configuration()
        //-----
        {
            InitializeComponent();

            // Set Default parameters
            EvButSetParameter(this, EventArgs.Empty);
        }

        //-----
        private void EvButSetParameter(object sender, EventArgs e)
        //-----
        {
            //
            // We read all the parameters from the text boxes.
            //
            m_parameters.Clear();

            var lines = m_tbResistors.Text.Split(new[] { "\r\n" }, StringSplitOptions.RemoveEmptyEntries);
            foreach (var line in lines)
            {
                double res = 0;
                if (double.TryParse(line, out res))
                {
                    m_parameters.Add(res);
                }
            }

            lines = m_tbInductances.Text.Split(new[] { "\r\n" }, StringSplitOptions.RemoveEmptyEntries);
            foreach (var line in lines)
            {
                double ind = 0;
                if (double.TryParse(line, out ind))
                {
                    m_parameters.Add(ind);
                }
            }

            lines = m_tbCapacities.Text.Split(new[] { "\r\n" }, StringSplitOptions.RemoveEmptyEntries);
            foreach (var line in lines)
            {
                double cap = 0;
                if (double.TryParse(line, out cap))
                {
                    m_parameters.Add(cap);
                }
            }

            lines = m_tbVoltFreq.Text.Split(new[] { "\r\n" }, StringSplitOptions.RemoveEmptyEntries);
            foreach (var line in lines)
            {
                double vol = 0;
                if (double.TryParse(line, out vol))
                {
                    m_parameters.Add(vol);
                }
            }

            lines = m_tbTiming.Text.Split(new[] { "\r\n" }, StringSplitOptions.RemoveEmptyEntries);
            foreach (var line in lines)
            {
                double time = 0;
                if (double.TryParse(line, out time))
                {
                    m_parameters.Add(time);
                }
            }
        }
    }
}
```



SerialRLC_configuration.cs code part 2:

```
//-----  
//  
// IEngineTrackSlave Interface  
//  
public void SetEngineTrack(IEngineTrackInfo engineTrack, int keyElement)  
//-----  
{  
    //  
    // This function is automatically called from BraceEngine build process  
    // to initialize the EngineTrack Slave (this Assembly).  
    //  
    m_engineTrack = engineTrack;  
    m_keyElement = keyElement;  
    // subscribe to include Transmit and Receive  
    m_engineTrack.Subscribe(this);  
    m_engineTrack.WriteLocalEvent("EngineTrack Subscribes SerialRLC_configuration Assembly (to local).",  
        m_keyElement,  
        TrackMsgType.Information);  
}  
//-----  
//  
// IEngineTrackSlave Interface  
//  
public void Transmit(int trackNumber, ref SEngineTrack data, Int64 curTicks, double curTime)  
//-----  
{  
    //  
    // This function is called from the Master (BraceEngine User C# code) to the Slave (this Assembly)  
    // to transmit data to the Master.  
    //  
    if (trackNumber == 100001)  
    {  
        // Master calls (receive) for some data from slave  
        int ind = 0;  
        data.dArray = new double[m_parameters.Count];  
        foreach (var param in m_parameters)  
        {  
            data.dArray[ind] = param;  
            ind++;  
        }  
        var localEvent = string.Format("Transmit Parameters from Assembly: Resistors {0} {1} {2} {3} {4}",  
            data.dArray[0],  
            data.dArray[1],  
            data.dArray[2],  
            data.dArray[3],  
            data.dArray[4]);  
        m_engineTrack.WriteLocalEvent(localEvent, m_keyElement, TrackMsgType.Information);  
        localEvent = string.Format("Transmit Parameters from Assembly: Inductances {0} {1} {2} {3} {4}",  
            data.dArray[5],  
            data.dArray[6],  
            data.dArray[7],  
            data.dArray[8],  
            data.dArray[9]);  
        m_engineTrack.WriteLocalEvent(localEvent, m_keyElement, TrackMsgType.Information);  
        localEvent = string.Format("Transmit Parameters from Assembly: Capacities {0} {1} {2} {3} {4}",  
            data.dArray[10],  
            data.dArray[11],  
            data.dArray[12],  
            data.dArray[13],  
            data.dArray[14]);  
        m_engineTrack.WriteLocalEvent(localEvent, m_keyElement, TrackMsgType.Information);  
        localEvent = string.Format("Transmit Parameters from Assembly: Voltage and Frequencies {0} {1} {2}",  
            data.dArray[15],  
            data.dArray[16],  
            data.dArray[17]);  
        m_engineTrack.WriteLocalEvent(localEvent, m_keyElement, TrackMsgType.Information);  
        localEvent = string.Format("Transmit Parameters from Assembly: Timing Ticks {0} {1} {2} {3}",  
            data.dArray[18],  
            data.dArray[19],  
            data.dArray[20],  
            data.dArray[21]);  
        m_engineTrack.WriteLocalEvent(localEvent, m_keyElement, TrackMsgType.Information);  
    }  
}  
//-----  
//  
// IEngineTrackSlave Interface  
//  
public void Receive(int trackNumber, SEngineTrack data, Int64 curTicks, double curTime)  
//-----  
{  
    //  
    // This function is called from the Master (BraceEngine User C# code) to the Slave (this Assembly)  
    // to receive data from the Master.  
    //  
    // not used  
}  
}
```

**Remarks**

- Derive the generated class additionally from IEngineTrackSlave (mandatory).
- Define functions SetEngineTrack(), Transmit(), Receive() (mandatory).
- Define the function call Subscribe(this) within SetEngineTrack() (mandatory).
- EvButSetParameter() is the Button event which also is used for internal initialization.
- The way to read the parameters and fill it up to the SEngineTrack data array is just an example.

6.1.8 BraceCalculation Assembly Simulation

After Build and Start the following curves are visible in the different scopes.

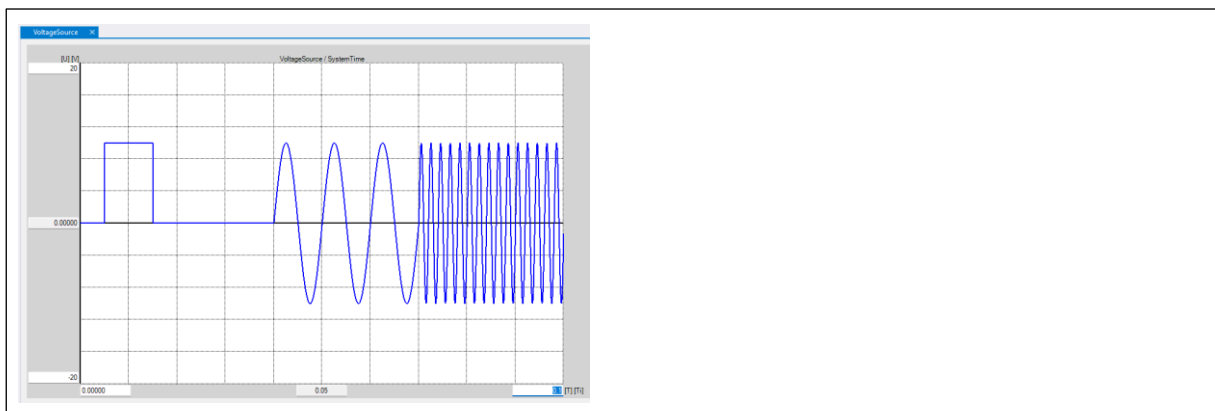


Figure 47 Voltage Source

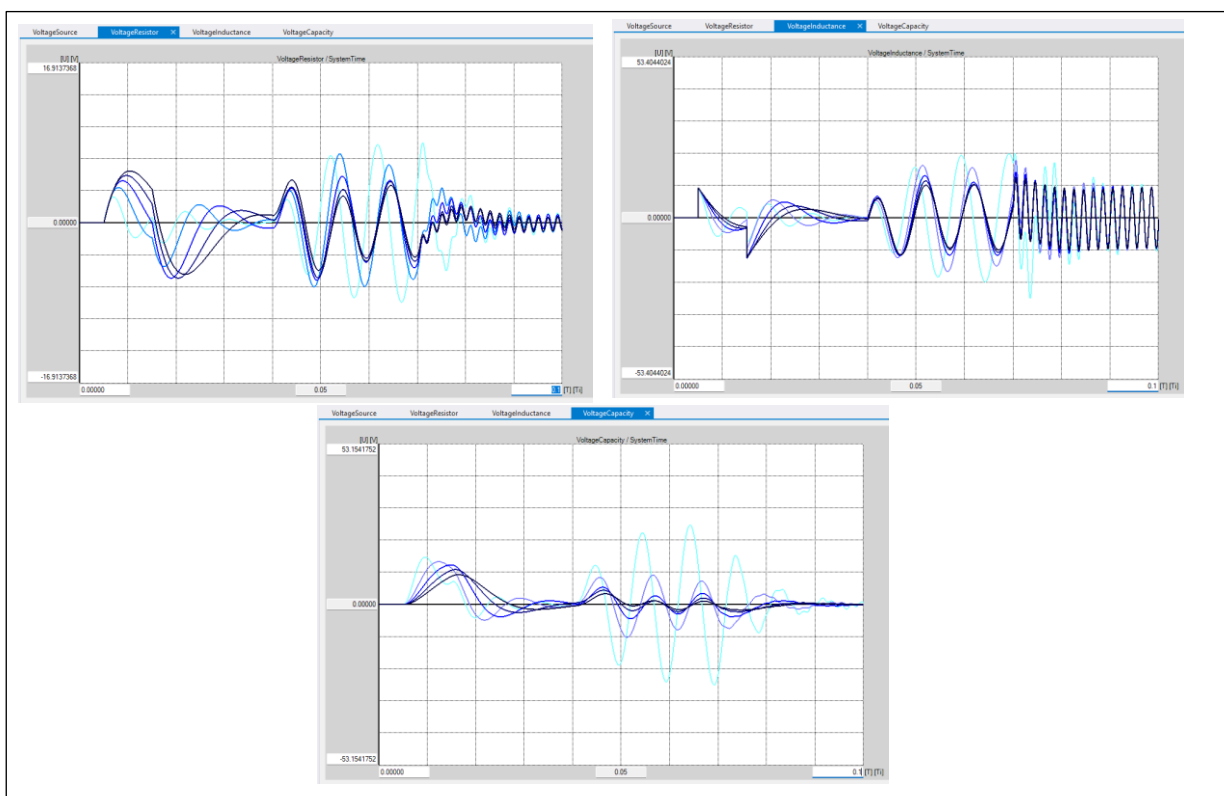


Figure 48 Voltages RLC

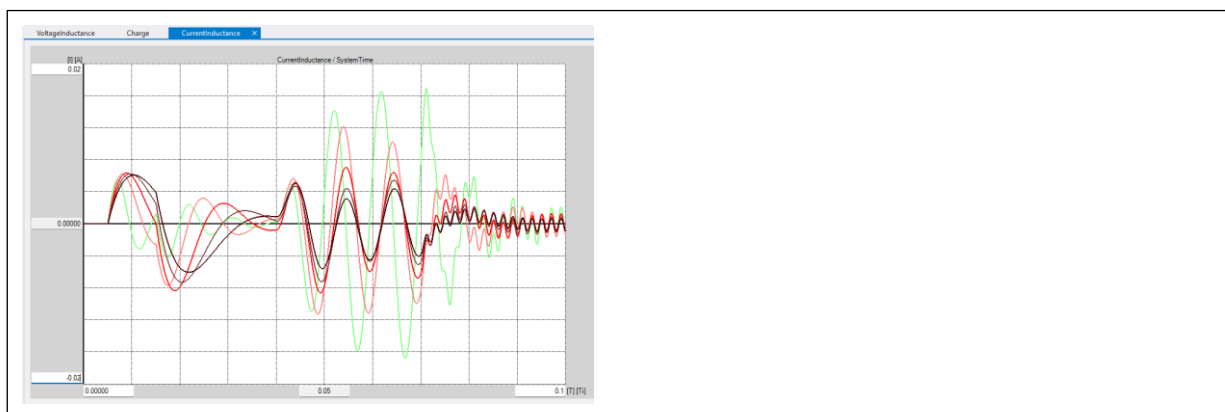


Figure 49 Current Inductance

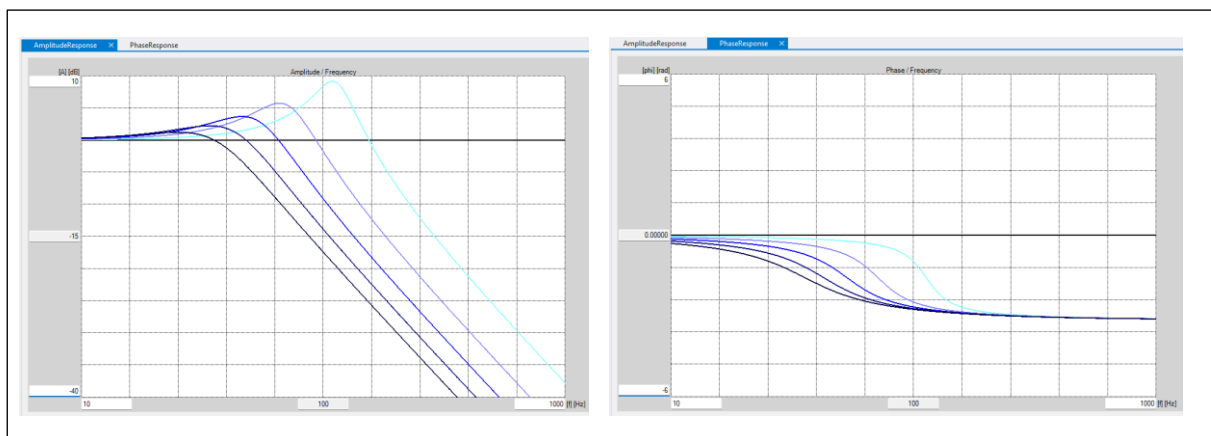


Figure 50 RLC Bode Diagram Curves