Limnor Studio – User's Guide

Part – III

Expressions

Contents

1 Introduction to Expressions

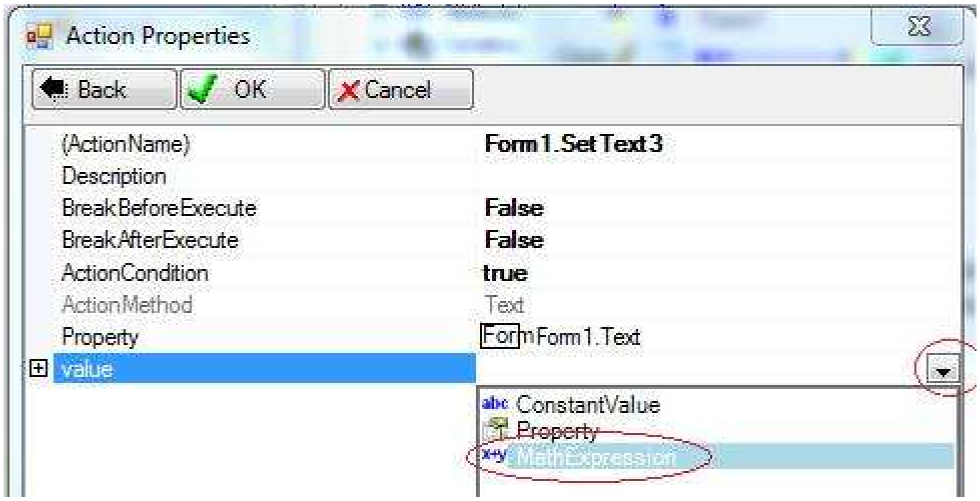
1.1 What are expressions

An expression represents a value formed by constant values, variables, and operators. For example, x + y is an expression formed by variable x and y, and an operator +.

A variable in an expression is mapped to a property in the software project to get value at runtime.

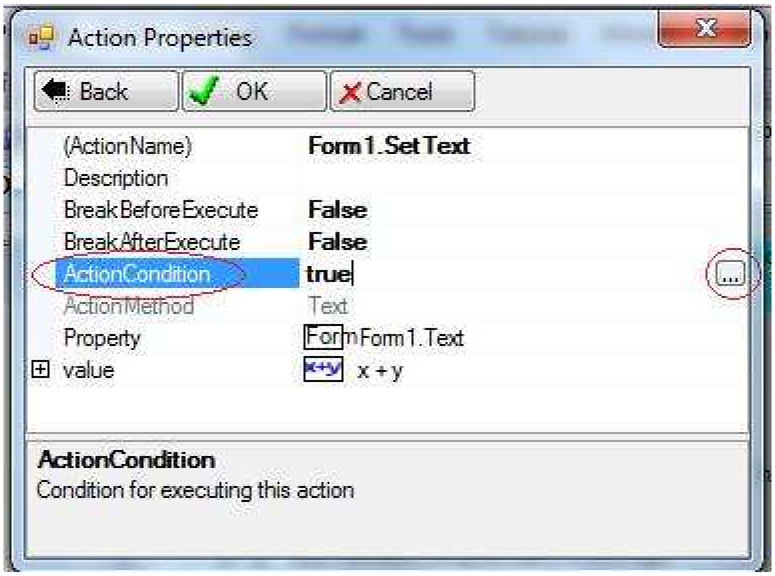
In Limnor Studio, expressions are used in actions to provide values for action parameters and set action conditions.

To use an expression to provide value for an action parameter, choose “MathExpression” for it:



(ActionName)	Form1.SetText3
Description	
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Text
Property	Form Form1.Text
value	

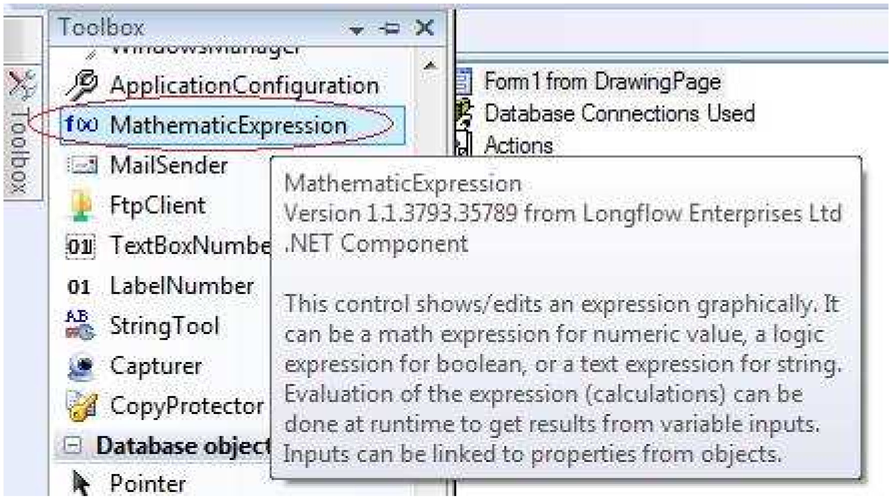
To use an expression to set action condition, click ... for ActionCondition of the action:



(ActionName)	Form1.SetText
Description	
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Text
Property	Form1.Text
⊞ value	x + y

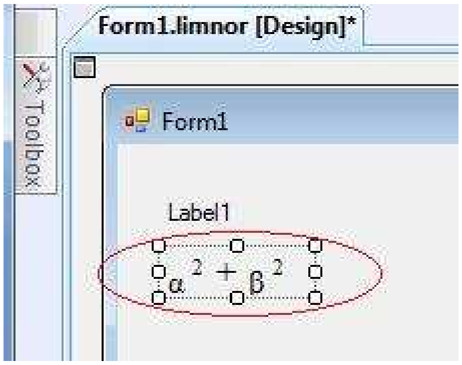
ActionCondition
Condition for executing this action

Limnor Studio also provides a Math Expression control for graphically displaying and editing expressions. It allows the user to do calculations at runtime using values from object properties.



MathematicExpression
Version 1.1.3793.35789 from Longflow Enterprises Ltd
.NET Component

This control shows/edits an expression graphically. It can be a math expression for numeric value, a logic expression for boolean, or a text expression for string. Evaluation of the expression (calculations) can be done at runtime to get results from variable inputs. Inputs can be linked to properties from objects.

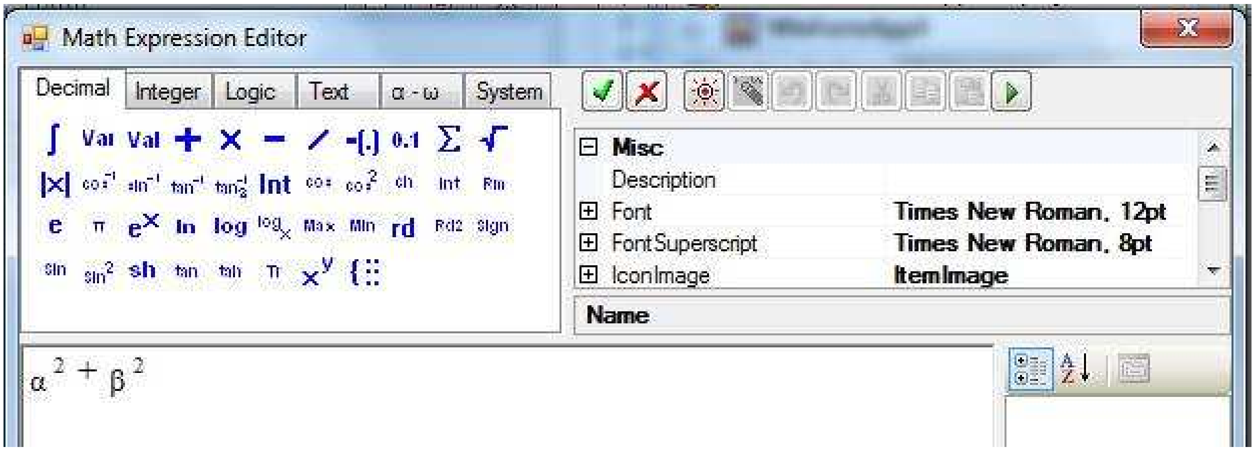


Limnor Studio supports 3 types of expressions: numeric expressions, logic expressions and text expressions.

- numeric expression – the result is a number
- logic expressions – the result is True or False
- text expressions – the result is a string

1.2 Create and edit expressions

Expression Editor is a visual formula editing tool:



Misc
Description
Font: Times New Roman, 12pt
FontSuperscript: Times New Roman, 8pt
IconImage: ItemImage

$\alpha^2 + \beta^2$

The formula is displayed and edited in its original graphic format. The user can intuitively create formula and does not need to learn new texture computer languages. Some examples of expressions:

$$\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$$

$$\int_0^{\Phi} \sin^2(\beta)\, d\beta$$

2 Expression Editor

2.1 Expression elements

An expression is constructed using elements. Limnor Studio ships with more than 80 expression elements. New elements can be added by copying the DLL files containing the elements to the Limnor Studio folder and listing the DLL files in an XML file named MathNode.XML. The MathNode.XML file shipped with Limnor Studio has the following contents:

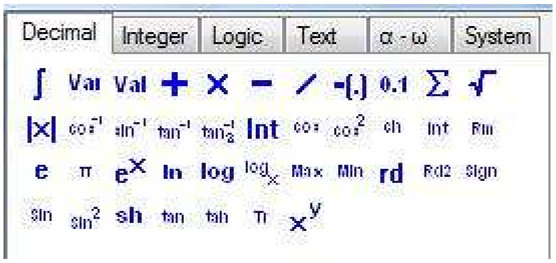
```
<MathNodeLib>
        <Lib>MathItem.dll</Lib>
</MathNodeLib>
```

The expression elements shipped with Limnor Studio are described below.

2.1.1 Numeric expression elements

Numeric expression elements are used to form mathematic expressions.

2.1.1.1 *Expression elements dealing with decimals*


Decimal
Integer
Logic
Text
α - ω
System
∫ Var Val + × − / -(.) 0.1 Σ √
|x| cos-1 sin-1 tan-1 tan2-1 Int cos cos2 ch Int Rm
e π ex ln log logx Max Min rd Rd2 Sign
sin sin2 sh tan tah Tr xy

+ Plus operator. computes the sum of its two operands

− Minus operator. subtracts the second operand from the first

× Multiplication operator. computes the product of its operands

/ Division operator. divides its first operand by its second

-(.) Negation operator. Multiply -1 to the selected math expression

0.1 constant decimal

Σ Sum operator.

√ square root

|x| get absolute value

∫ integration

Var A variable

Val A constant decimal value. The value type may be changed.

$\cos^{-1}$ The angle whose cosine is the specified number

$\sin^{-1}$ The angle whose sine is the specified number

$\tan^{-1}$ The angle whose tangent is the specified number

$\tan_2^{-1}$ The angle whose tangent is the quotient of two specified numbers

Int The largest integer less than or equal to the specified number

Int The smallest integer greater than or equal to the specified number

cos The cosine of the specified angle

$\cos^2$ The square of the cosine of the specified angle

ch The hyperbolic cosine of the specified angle

Rm The remainder resulting from the division of a specified number by another specified number

e The natural logarithmic base, specified by the constant, e

π The ratio of the circumference of a circle to its diameter, specified by the constant π

e^x e raised to the specified power

ln The base e logarithm of a specified number

log The base 10 logarithm of a specified number

$\log_x$ The logarithm of a specified number, for a specified base

Max The larger of two specified numbers

Min The smaller of two specified numbers

rd Rounds a value to the nearest integer

Rd2 Rounds a value to the nearest integer or specified number of decimal places.

Sign A value indicating the sign of a number

Sin The sine of the specified angle

$\sin^2$ The square of the sine of the specified angle

sh The hyperbolic sine of the specified angle

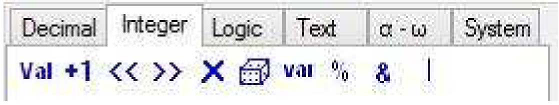
tan The tangent of the specified angle

tah The hyperbolic tangent of the specified angle

Tr Calculates the integral part of a number

x^y A specified number raised to the specified power

2.1.1.2 Expression elements dealing with integers



Val A constant integer value. The value type may be changed.

+1 Increase the number by 1

<< Shift the number left by given bits

>> Shift the number right by given bits

X Produces the full product of two 32-bit numbers, and generates a 64-bit number

Generate a random number in a specified range

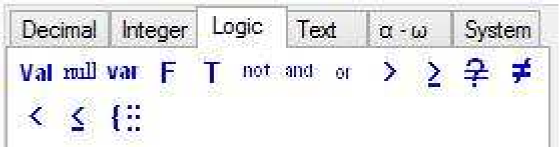
var An integer variable

% Computes the remainder of dividing its first operand by its second

& Bitwise 'and' operator

| Bitwise 'or' operator

2.1.2 Logic expression elements



A logic expression evaluates to a Boolean result.

var A Boolean variable

null A null value

F A Boolean false

T A Boolean true

not Logic 'not' operator

and Logic 'and' operator

or Logic 'or' operator

> Logic 'greater than' operator

≥ Logic 'greater than or equal to' operator

≟ Logic 'equal' operator

≠ Logic 'not equal' operator

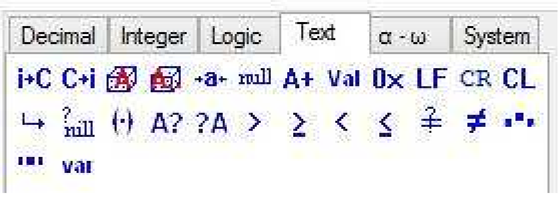
< Logic 'less than' operator

≤ Logic 'less than or equal to' operator

{:: Calculations with conditions

Note that if comparison for string then case-sensitive comparison is performed. To use case-sensitivity options, use operators for text expressions.

2.1.3 Text expression elements


Decimal
Integer
Logic
Text
α - ω
System
i•C C•i •a• null A+ Val 0x LF CR CL
?null (•) A? ?A > ≥ < ≤ ≟ ≠
var

These elements deal with strings.

0x It represents an ASCII character. Usually it is used for an unprintable character.

LF It represents a Line Feed character. It is ASCII character 0x0A

CR It represents a Carriage Return character. It is ASCII character 0x0D

CL It represents a Carriage Return and a Line Feed. It is a pair of ASCII characters 0x0D0A. Windows text files usually use it as line separator

↳ It represents a tab character. It is ASCII character 0x09

?null Tests whether the specified string is null or empty

(•) Tests whether the second string is part of the first string

A? Tests whether the first string starts with the second string

?A Tests whether the first string ends with the second string

> Tests whether the first string is greater than the second string

≥ Tests whether the first string is greater than or equal to the second string

≟ Tests whether the first string is equal to the second string

≠ Tests whether the first string is not equal to the second string

"" A constant string

var A string variable

Val It is a constant value. The value type may be changed.

null A null value

It generates a random string

+a+ It removes whitespaces from both sides of a string

i+C It converts an integer to a character. For example, 65 is converted to A, 66 is converted to B, etc.

C+i It converted a character to its ASCII integer. For example, A is converted to 65, B is converted to 66, etc.

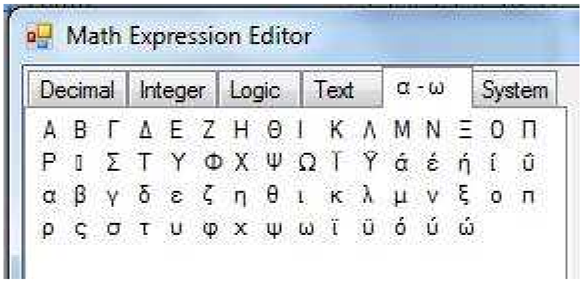
A+ Concatenate two strings together to form a new string

It generates a random character

It gets a sub-string of a string by specifying start position and length.

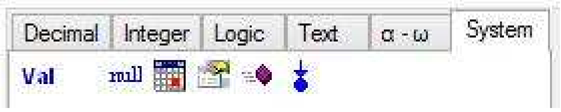
2.1.4 Greek symbols

Greek symbols are commonly used in mathematic formulas. You can use Greek symbols in any variable names and subscription names.


Math Expression Editor
Decimal
Integer
Logic
Text
α - ω
System
Α Β Γ Δ Ε Ζ Η Θ Ι Κ Λ Μ Ν Ξ Ο Π
Ρ Σ Τ Υ Φ Χ Ψ Ω Ϊ Ϋ ά έ ή ί ΰ
α β γ δ ε ζ η θ ι κ λ μ ν ξ ο π
ρ ς σ τ υ φ χ ψ ω ϊ ϋ ό ύ ώ

2.1.5 System elements

Systems elements deal with programming elements.



null It represents a null value. It can be used in a logic expression to test whether a variable is null or not.

It represents current date and time

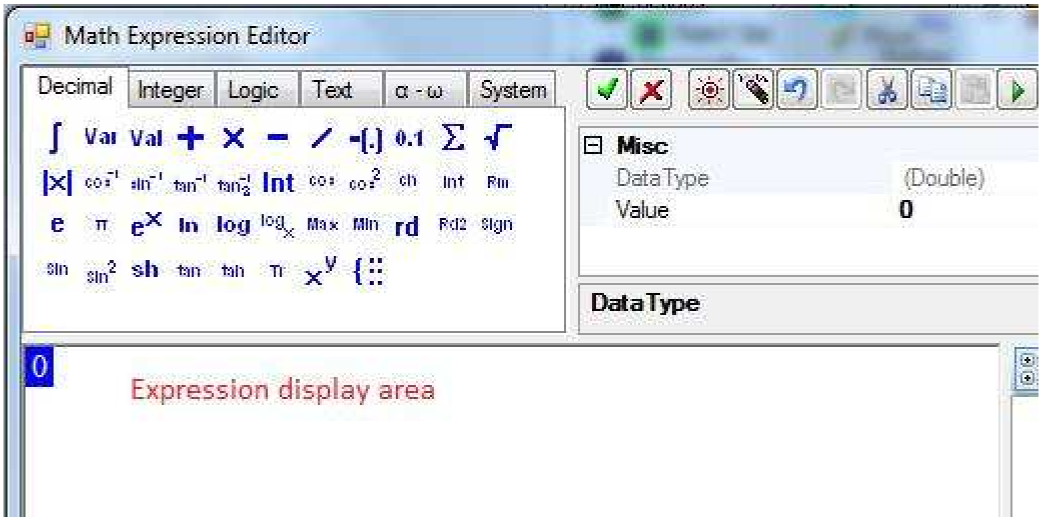
It is a reference to a property of an object. It can also represent an object, a parameter, or a variable.

It represents a return value of a method execution.

It represents an input value of an action. The input of an action is the output of the last action.

2.2 Use Keyboard

When the input focus is at the expression display area, keyboard can be used to edit the expression. If you press a key without a response then the expression display area does not have input focus. Click the expression display area to give it input focus.



Misc
DataType (Double)
Value 0

DataType

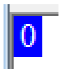
0

Selected element is highlighted. When a key is pressed the editing result depends on which element is selected. It is very important to pay attention to which parts of an expression are highlighted while you modify the expression.

2.2.1 Basic operator keys: + - * /

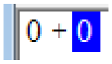
When one of these 4 keys is pressed, a new operant 0 is added to the selected element.

Suppose initially the expression is a 0:



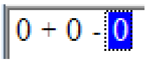
0

Press +. A new operant is added with +:



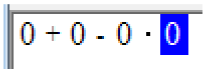
0 + 0

Press -. A new operant 0 is added with -:



0 + 0 - 0

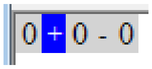
Press *. A new operant 0 is added with multiplication:

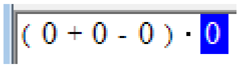


0 + 0 - 0 · 0

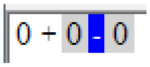
The new 0 multiplies the last 0 because the last 0 is selected (highlighted).

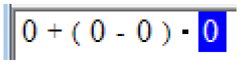
If we press * when the whole expression is selected then the result becomes


0 + 0 - 0

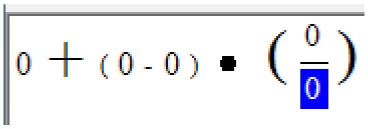

(0 + 0 - 0) · 0

If we press * when the minus expression is selected then the result becomes

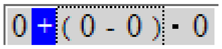

0 + 0 - 0

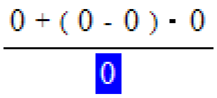

0 + (0 - 0) · 0

Now press / the expression becomes:

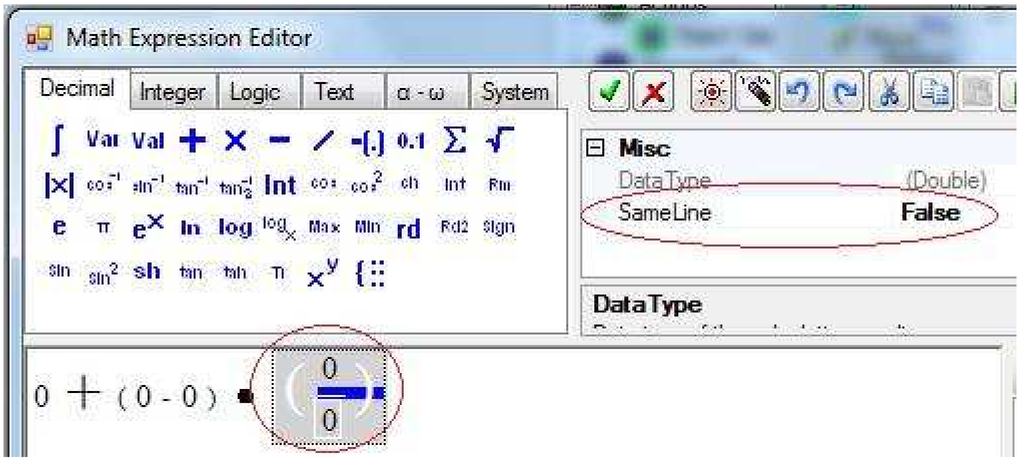

$0 + (0 - 0) \cdot \left(\frac{0}{0}\right)$

If we press / when the plus expression is selected then the result becomes


0 + (0 - 0) · 0


$\frac{0 + (0 - 0) \cdot 0}{0}$

When an element is selected the properties of the element are displayed. Changing properties may affect the display of the expression. Let’s select the division element:


Math Expression Editor
Decimal
Integer
Logic
Text
α - ω
System
∫ Var Val + × − / -(.) 0.1 Σ √
|x| cos⁻¹ sin⁻¹ tan⁻¹ tan⁻¹₂ Int cos cos² ch Int Rm
e π eˣ ln log logₓ Max Min rd Rd2 Sign
sin sin² sh tan th π xʸ {::

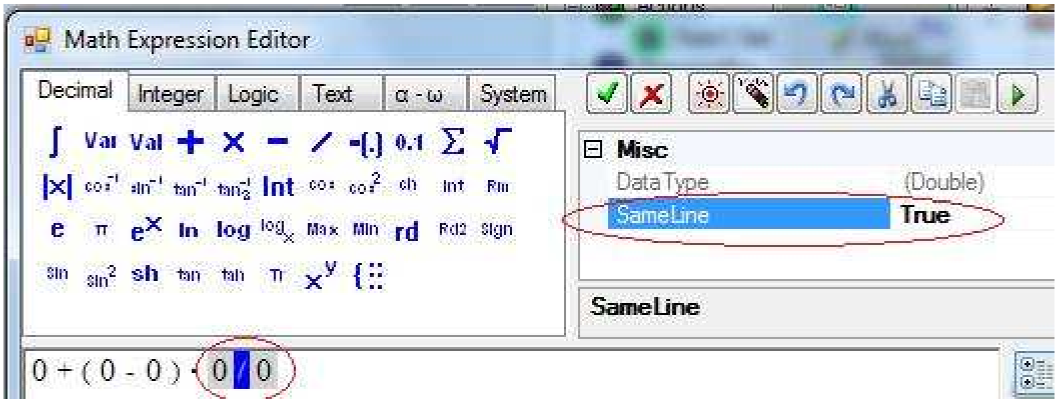
Misc

DataType	(Double)
SameLine	False

DataType

$0 + (0 - 0) \cdot \left(\frac{0}{0}\right)$

Change property SameLine to True. The display of the expression becomes:



Misc

DataType (Double)

SameLine True

SameLine

0 + (0 - 0) · 0 / 0

2.2.2 Number keys

When a number key is pressed while a number element or a variable element is selected, the number being pressed replaces the selected element.

While the last 0 is selected 0 + (0 - 0) · 0 / 0 , press 2, it becomes 0 + (0 - 0) · 0 / 2 .

Keep typing numeric keys to enter the number 0 + (0 - 0) · 0 / 2.345

2.2.3 Alphabet keys

When an alphabet key is pressed while a number element or a variable element is selected, the selected element is replaced with a variable element using the alphabet key as the first letter of the variable name.

Initially: 0 + (0 - 0) · 0 / 2.345 . Press t: 0 + (0 - 0) · t / 2.345 . Keep typing op: 0 + (0 - 0) · top / 2.345

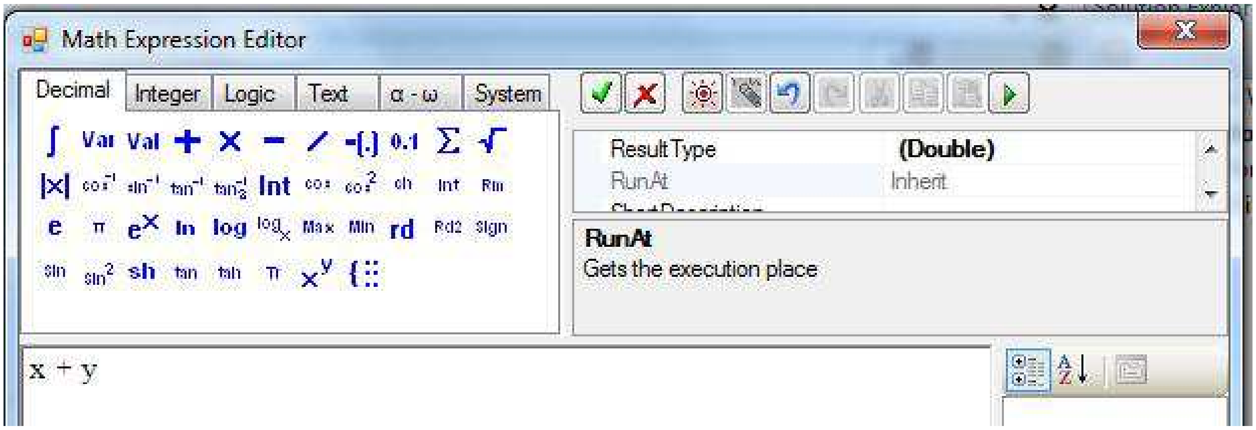
2.2.4 Backspace and Delete keys

Backspace and Delete keys can be used to delete the selected element.

Initially: 0 + (0 - 0) · top / 2.345 . Press Backspace or Delete key: 0 + 0 · top / 2.345

2.3 Replace operators

Suppose we have an expression x + y.

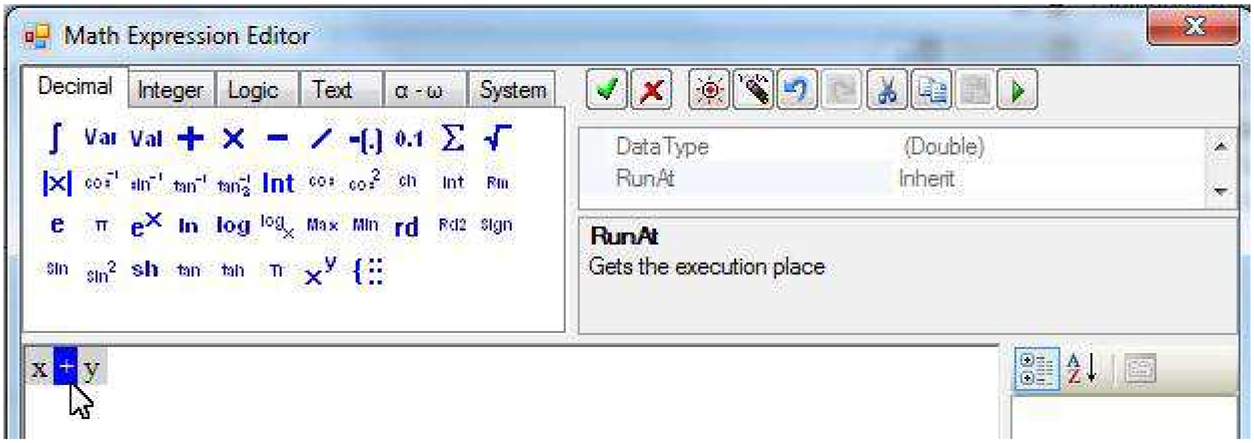


Result Type	**(Double)**
RunAt	Inherit

RunAt
Gets the execution place

x + y

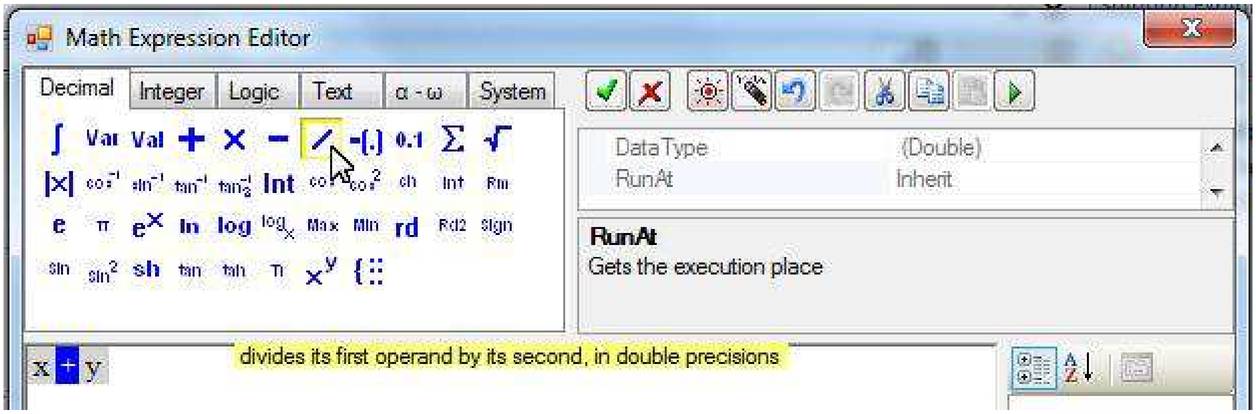
If we want to modify it to be x / y then we need to select "+" and click "/".



DataType	(Double)
RunAt	Inherit

RunAt
Gets the execution place

x + y

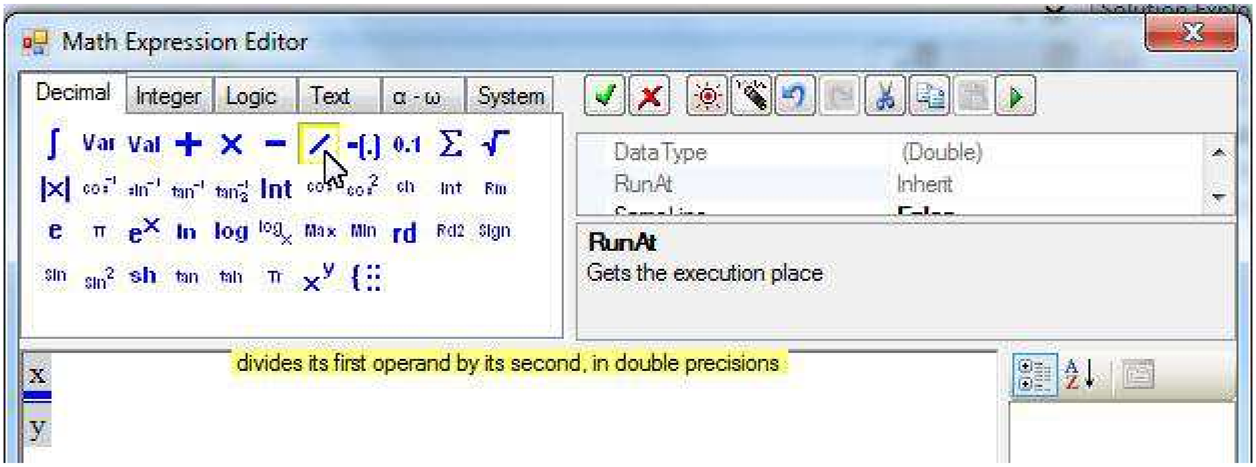


DataType	(Double)
RunAt	Inherit

RunAt
Gets the execution place

x + y

divides its first operand by its second, in double precisions

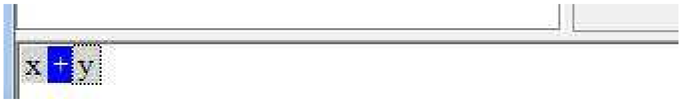


DataType	(Double)
RunAt	Inherit

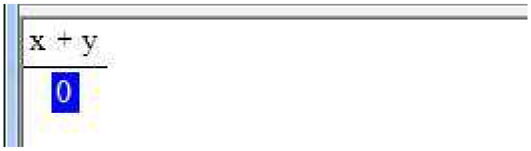
RunAt
Gets the execution place

divides its first operand by its second, in double precisions

If you do not want to replace the selected operator but to apply the new operator to the selected operator then you need to use the keyboard. For example, while “+” operator is selected:



Press “/” key, the “/” operator is applied to “x + y” and the expression becomes (x+y)/0:



3 Variables in Expressions

3.1 Create a variable

Select a number, type the first letter of the variable to create the variable. In previous section, we created a variable named top in this way.

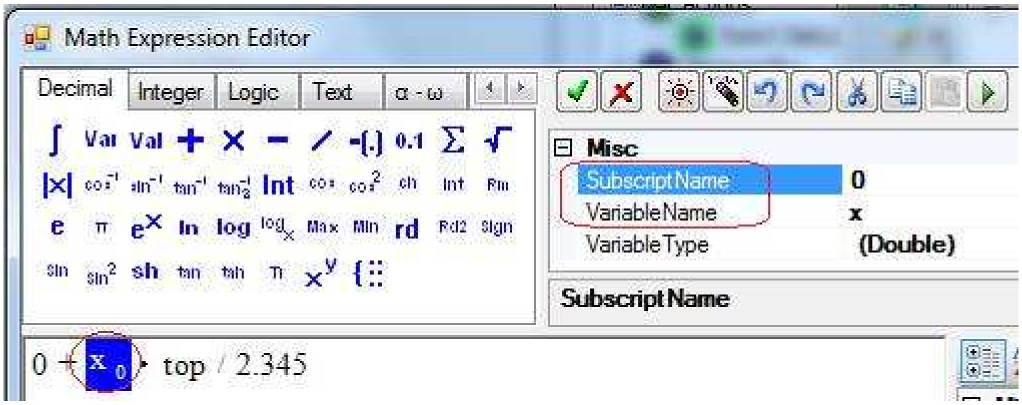
Select an element and click variable icon Var. A new variable is created replacing the selected element.

Initially: 0 + (3 - 1) · top / 2.345 . Click icon Var. The expression becomes 0 + x · top / 2.345 .

3.2 Name and subscript

A variable name cannot start with a number. Subscript of a variable name can start with a number. A variable is uniquely identified by its name and subscript.

Name and subscript of a variable can be changed when the variable is selected:



SubscriptName 0

VariableName x

VariableType (Double)

SubscriptName

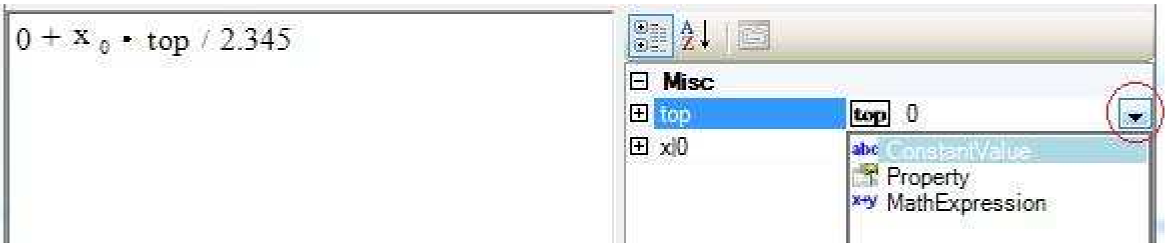
$0 + x_0 \cdot top / 2.345$

3.3 Provide values to variables

To provide value to a variable we may use the following 3 ways:

- Give it a constant value
- Map the variable to a property in the software within the scope where the expression is used
- Create another expression for it

In the Expression Editor, all variables are listed on the right side of the display area. We may provide values to variables in that list. Select a variable in the list, click ▾



$0 + x_0 \cdot top / 2.345$

top: top 0

These 3 ways of providing values are exactly the same as what are described in Part II of the User's Guide in chapter "Use Parameters in Actions". See http://www.limnor.com/support/Limnor%20Studio%20-%20User%20Guide%20-%20Part%20II.pdf.

4 Insert Elements into Expressions

We have seen that by pressing keys +, -, * and \, the expression can grow with new terms. The default new term is a number 0. We may change it to a variable or other element.

Other elements can be added to the expression by clicking the corresponding icons.

4.1 Insert function elements

A function element has one or more parameters.

Click a function element icon, a new function element will be created in the place of the selected element. The selected element will become the first parameter of the function.

The following examples show the process.

Initially: $y + x_0 \cdot top / 2.345$

If we click cos then it becomes $\cos(y) + x_0 \cdot top / 2.345$

If we click $\tan^{-1}_{2}$ then it becomes $\tan^{-1}(y, 0) + x_0 \cdot top / 2.345$

4.2 Insert logic operator elements

A logic operator element operates on two logic elements:

{first logic element} {logic operator} {second logic element}

Click a logic operator element icon, a new logic operator element will be created in the place of the selected element. The selected element will become the first logic element for the logic element. A new logic variable element will be created and used as the second logic element for the logic element.

The following examples show the process.

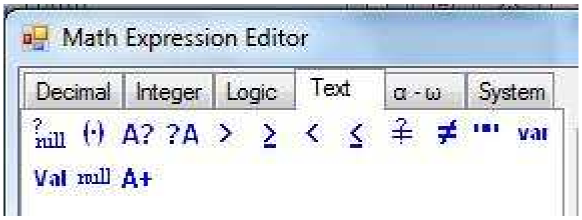
Initially: g_0

If we click > then it becomes $g_0 > x$. Note that g_0 becomes the first logic element for operator > and a new logic variable x becomes the second logic element for operator >.

Suppose we select g_0 again: $g_0 > x$. Click or. The expression becomes $(g_0 \text{ OR } x) > x$

4.3 Text testing operators

Text testing operators give logic Boolean results by doing various text testing. You may specify culture information to be used for doing testing.



4.3.1 Check empty

Icon: $\overset{?}{null}$

Example: $IsEmpty(s)$

This expression evaluates to Boolean True if variable s is null or an empty string.

4.3.2 Check containment

Icon: (·)

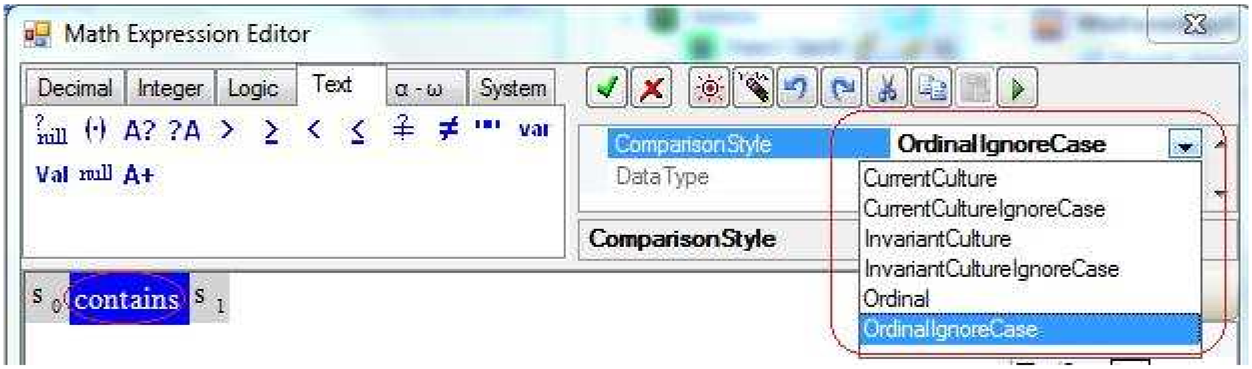
Example:

s_0 contains s_1

This expression evaluates to Boolean True if variable s_1 is part of variable s_0. For example, if s_1 is "Hello World" and s_0 is "llo" then the result is True. If s_0 is "llP" then the result is False.

By default, if s_0 is "LL" then the result is still True because "LL" matches "ll" if ignoring the case.

By default the ComparisonStyle property for the operator (·) (contains) is "OrdinalIgnoreCase":



Math Expression Editor

ComparisonStyle: OrdinalIgnoreCase

s_0 contains s_1

These comparison styles are supported in standalone projects:

- CurrentCulture – The comparison is done using the current culture of the computer, case-sensitively.
- CurrentCultureIgnoreCase – The comparison is done using the current culture of the computer, case-insensitively
- InvariantCulture – The comparison is done culture-independently (in English without regarding countries and regions) and case-sensitively
- InvariantCultureIgnoreCase – The comparison is done culture-independently and case-insensitively
- Ordinal – The comparison is done byte-by-byte and case-sensitively
- OrdinalIgnoreCase – The comparison is done byte-by-byte and case-insensitively

For web client operations, the culture specification is ignored, only case sensitivity option is used.

4.3.3 Checking beginning of text

Icon: A?

Example:

s_0 startsWith s_1

This expression evaluates to Boolean True if variable s_1 is the beginning part of variable s_0. For example, if s_1 is "Hello World" and s_0 is "He" then the result is True. If s_0 is "her" then the result is False.

This operator also supports comparison style.

4.3.4 Checking ending of text

Icon: ?A

Example: s_0 endsWith s_1

This expression evaluates to Boolean True if variable s_1 is the ending part of variable s_0. For example, if s_1 is "Hello World" and s_0 is "ld" then the result is True. If s_0 is "idl" then the result is False.

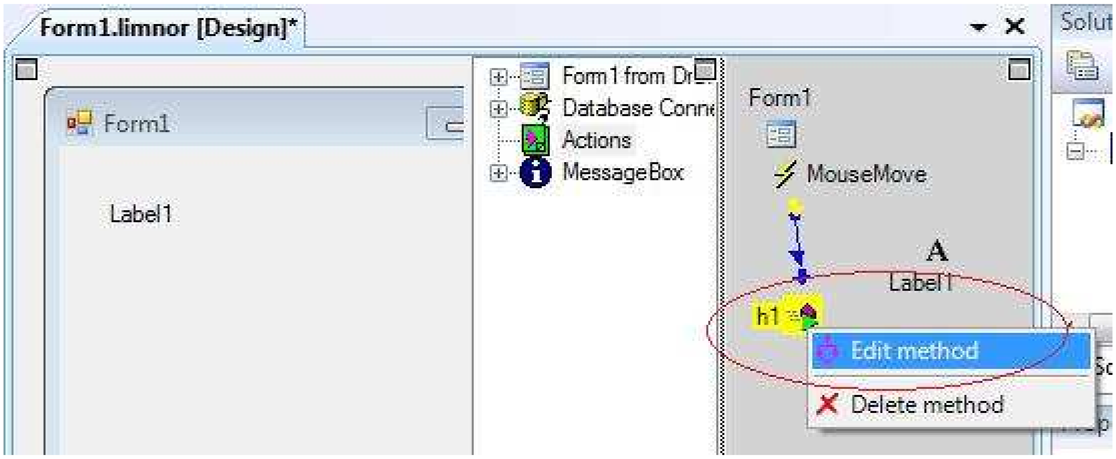
This operator also supports comparison style.

5 An Expression Example

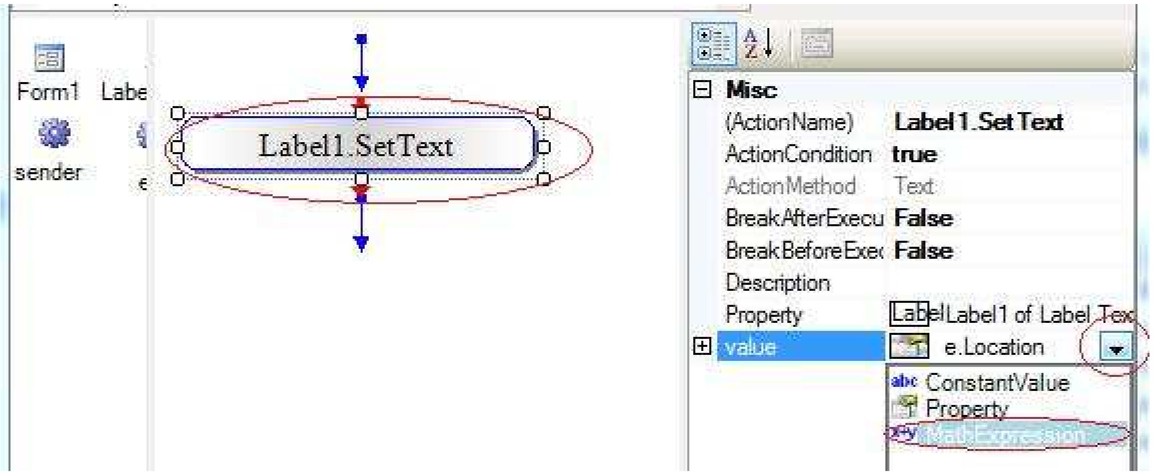
5.1 Sample task

In Part II of the User's Guide, we created a sample of showing mouse pointer location when mouse is moving. We may change it to show a distance between the mouse pointer and the center of the form. This involves creating a math expression to calculate the distance.

Open that sample project, open class Form1. Right-click the event-handler method icon in the Event Path; choose "Edit method":



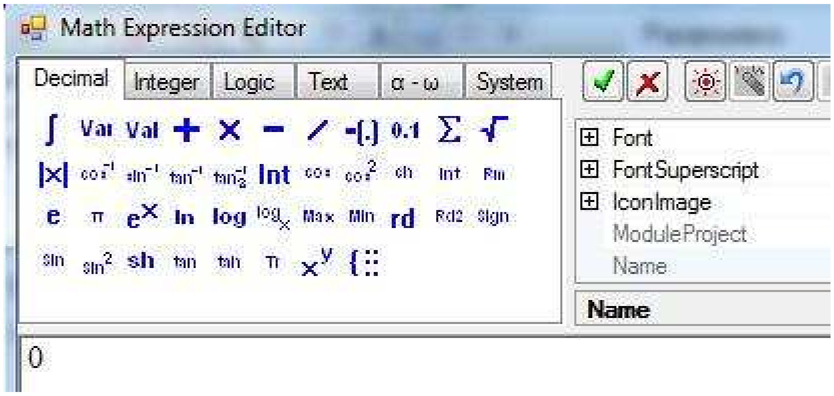
Select the action Label1.SetText. In Part II of the User's Guide, the value parameter of this action was set to the event parameter e.Location which is the mouse pointer location. For this sample, we change it to use a math expression to calculate the distance from the mouse pointer location to the center of the form. Click ▾, choose "MathExpression":



Misc

(ActionName)	Label1.SetText
ActionCondition	true
ActionMethod	Text
BreakAfterExecu	False
BreakBeforeExec	False
Description	
Property	Label1 of Label Tex
value	e.Location

The Expression Editor appears for us to create the expression.



5.2 Expression creation process

This is the math expression for calculating the distance between point (x_1, y_1) and point (x_0, y_0):

$$\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$$

Using the Expression Editor we may create this expression in a few second. We will describe the process of creating this expression. Note that you do not have to follow exactly the process used here to create this math expression. You may have your own preferences and habits. Pay attention to the highlighting in each step below.

Press -, the expression becomes $0 - \boxed{0}$

Press x, it becomes $0 - \boxed{x}$

Click 0, it becomes $\boxed{0} - x$

Type x, it becomes $\boxed{x} - x$

Click -, It becomes $x - x$

Click x^y, it becomes $(x - x)^2$

Press +, it becomes $(x - x)^2 + 0$

Press y, it becomes $(x - x)^2 + y$

Press -, it becomes $(x - x)^2 + y - 0$

Press y, it becomes $(x - x)^2 + y - y$

Click -, it becomes $(x - x)^2 + y - y$

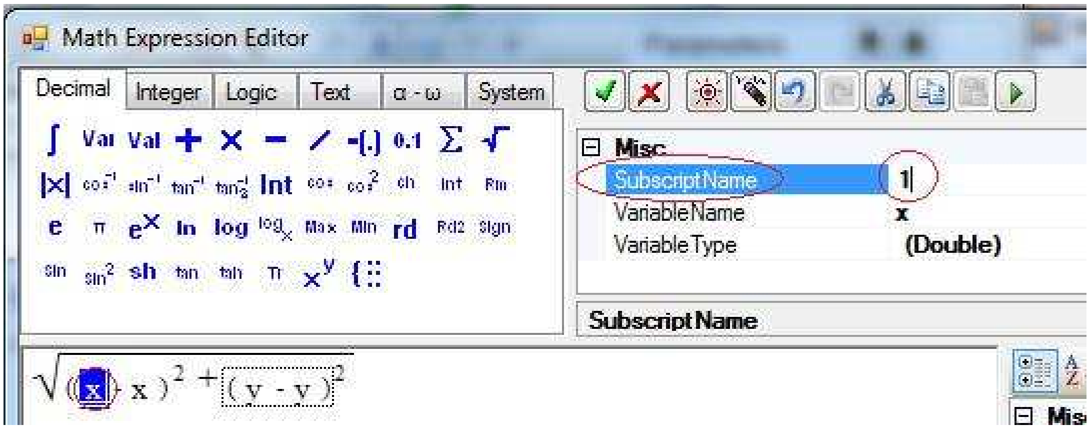
Click x^y, it becomes $(x - x)^2 + (y - y)^2$

Click +, it becomes $(x - x)^2 + (y - y)^2$

Click √, it becomes $\sqrt{(x - x)^2 + (y - y)^2}$

Click the first x, it becomes $\sqrt{(x - x)^2 + (y - y)^2}$

Set SubscriptName to 1:

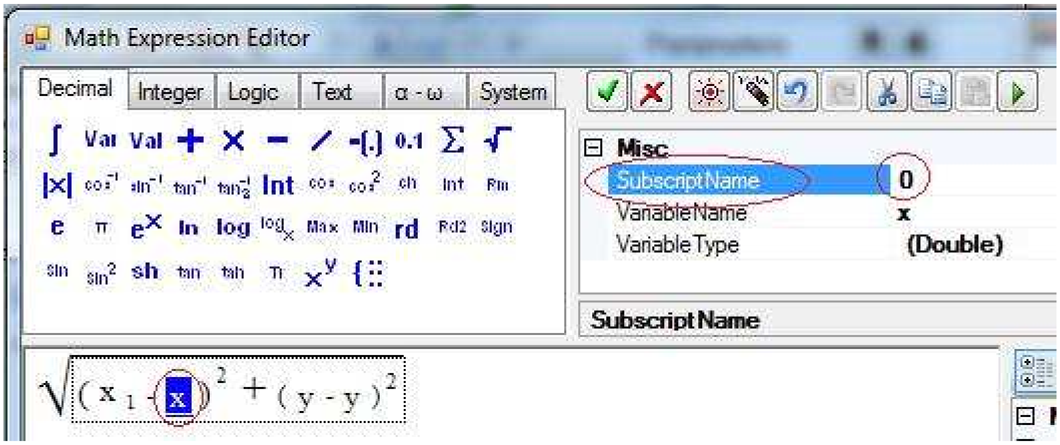


SubscriptName	1
VariableName	x
VariableType	(Double)

SubscriptName

$\sqrt{(x - x)^2 + (y - y)^2}$

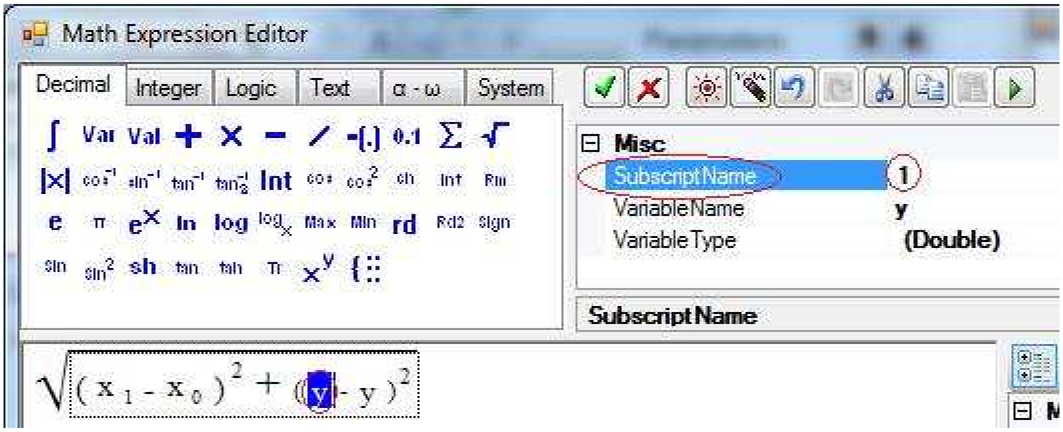
Click the second x and set its SubscriptName to 0:



SubscriptName 0
VariableName x
VariableType (Double)

$\sqrt{(x_1 - x)^2 + (y - y)^2}$

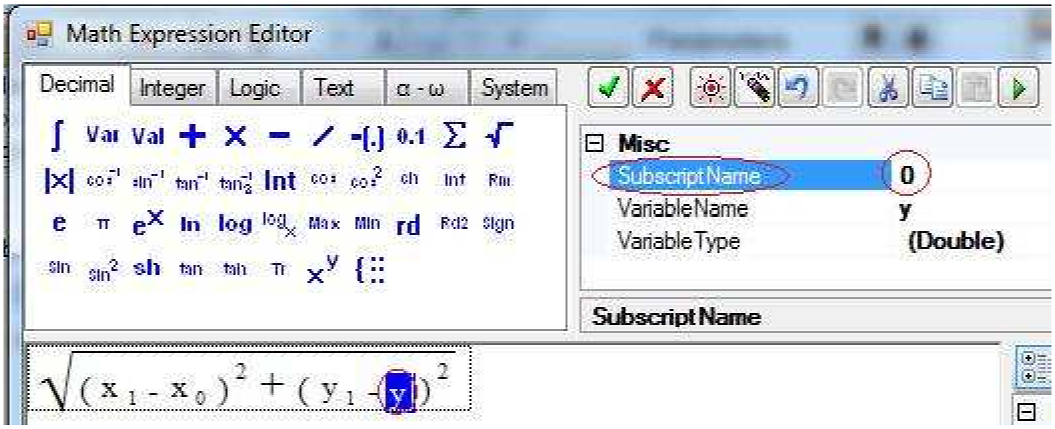
Select the first y and set its SubscriptName to 1:



SubscriptName 1
VariableName y
VariableType (Double)

$\sqrt{(x_1 - x_0)^2 + (y - y)^2}$

Click the second y and set its SubscriptName to 0:



SubscriptName 0
VariableName y
VariableType (Double)

$\sqrt{(x_1 - x_0)^2 + (y_1 - y)^2}$

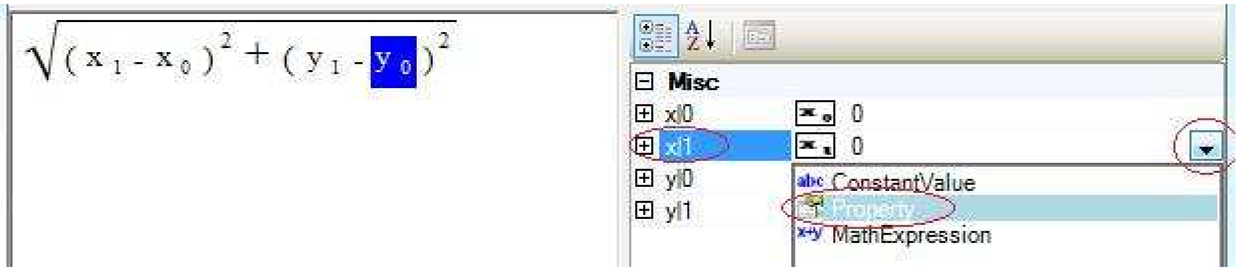
5.3 Provide values to variables

Now we created the expression for calculating distance between two points:

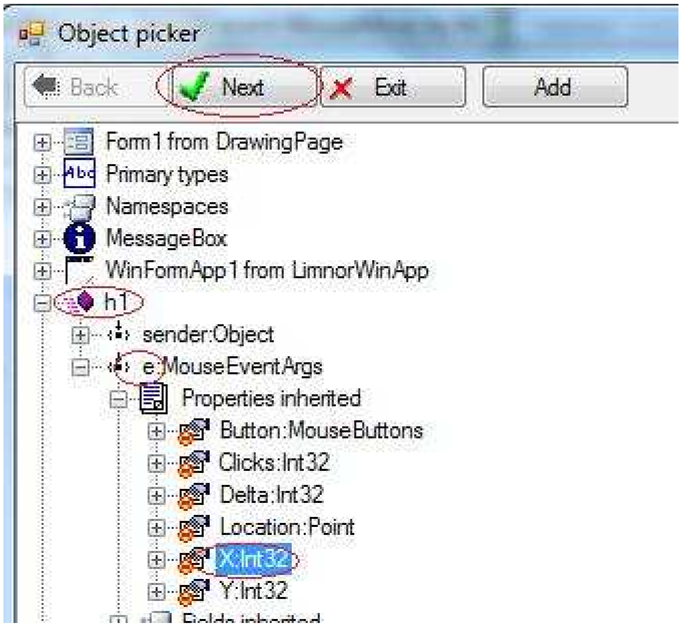
$$\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$$

Here, point (x_1, y_1) is the mouse pointer coordinates and point (x_0, y_0) is the center of the form. We need to enter these values through the variable list.

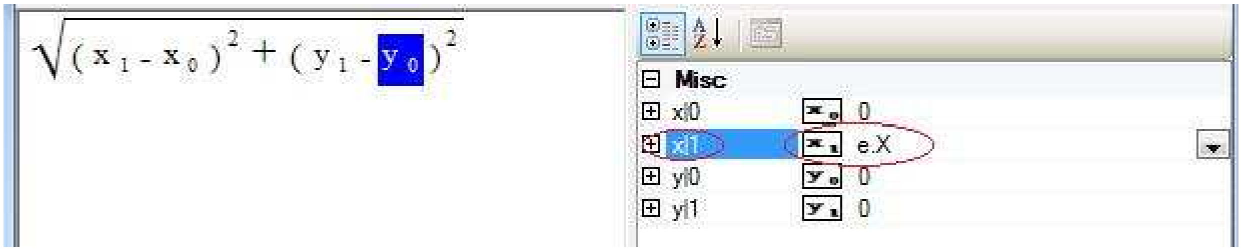
In the variable list, for x_1, click ▾ and select Property to get mouse location:



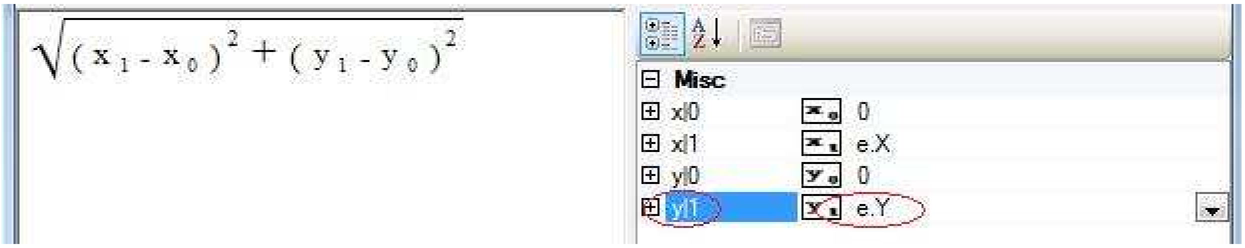
Under the event-handler method, find the mouse pointer location, select X, click Next:



The X coordinate of the mouse pointer location is provided to x_1:



In the similar process, the Y coordinate of the mouse pointer location is provided to y_1:

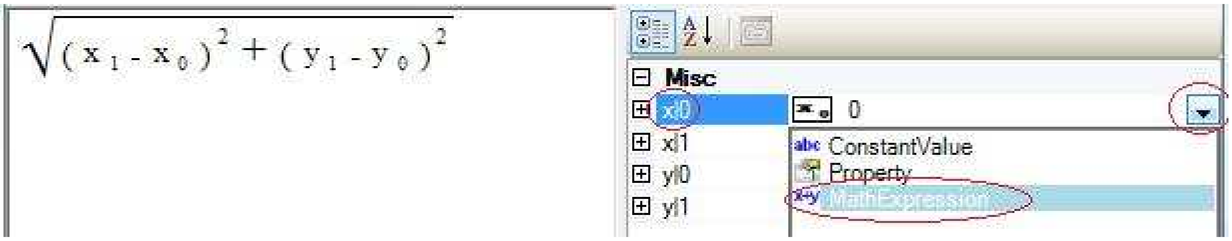


Point (x_0, y_0) is the center of the form.

x_0 can be calculated by a math expression {Width of the form}/2.

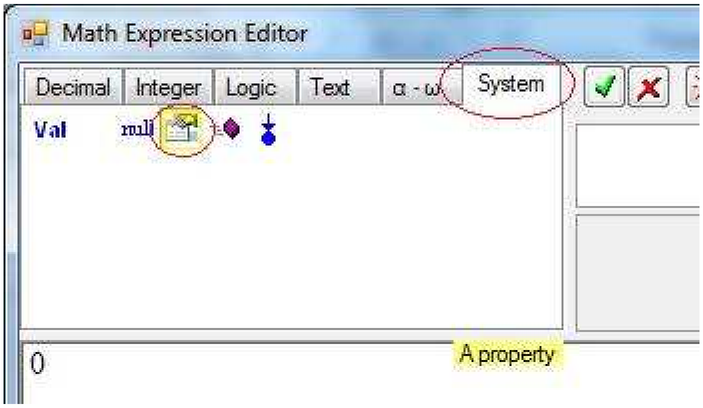
y_0 can be calculated by a math expression {Height of the form}/2.

In the variable list, for x_0, click ▾ and select MathExpression to get mouse location:

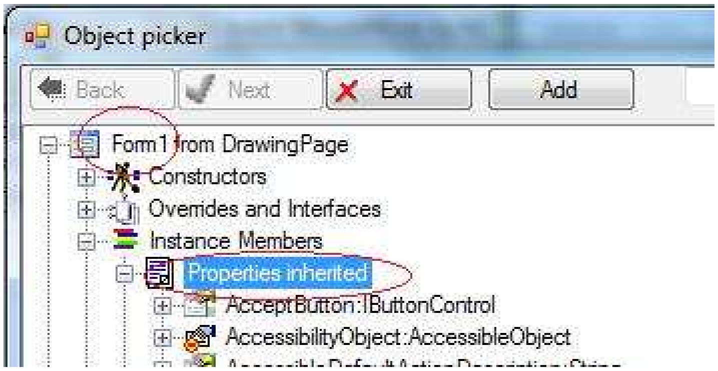


The Expression Editor appears.

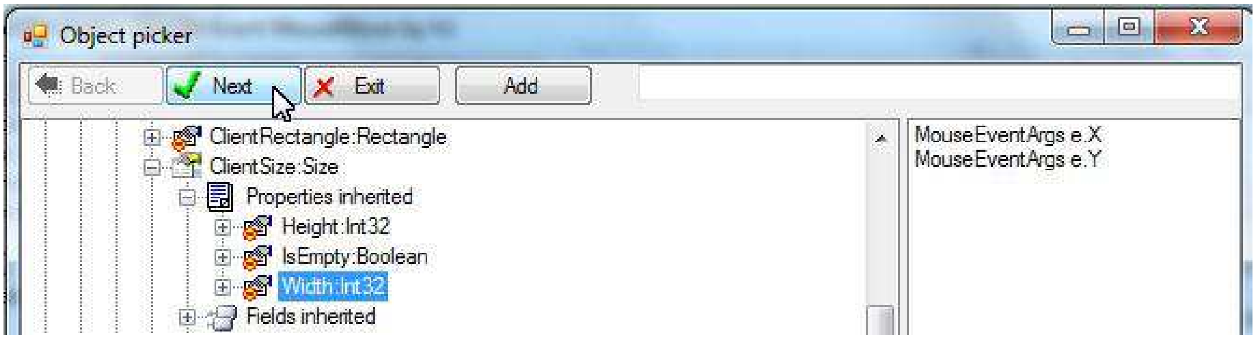
Click Property icon in the System tab because we want to use the Width property of the form:



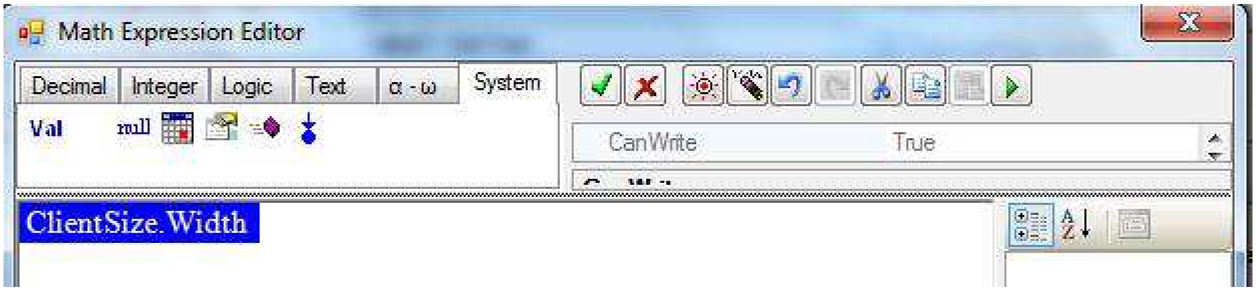
Expand the node “Properties inherited” to find the Width property of the form:



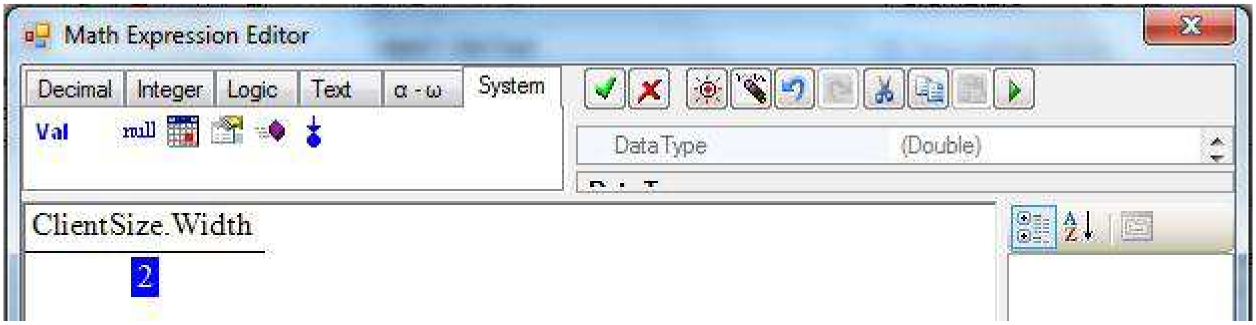
Select the Width property under ClientSize. Click Next:



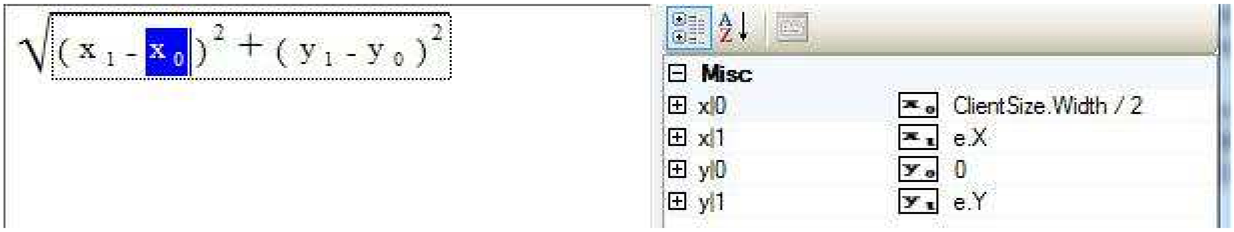
ClientSize.Width appears in the expression:



Type /2:



Click ✓.This expression is assigned to x_0:



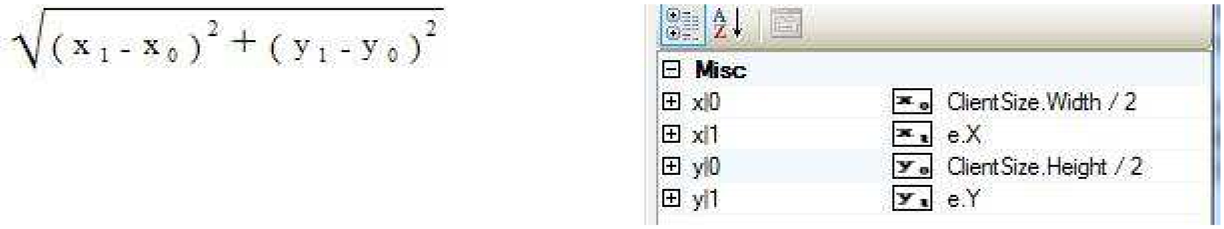
$$\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$$

Misc

x0	ClientSize.Width / 2
x1	e.X
y0	0
y1	e.Y

Here we see that a math expression is used to provide value to a variable in another math expression. The expression is formed by property "ClientSize.Width" of the form divided by 2.

In a similar way, we may use an expression to provide value to y_0. The expression is formed by property "ClientSize.Height" of the form divided by 2:

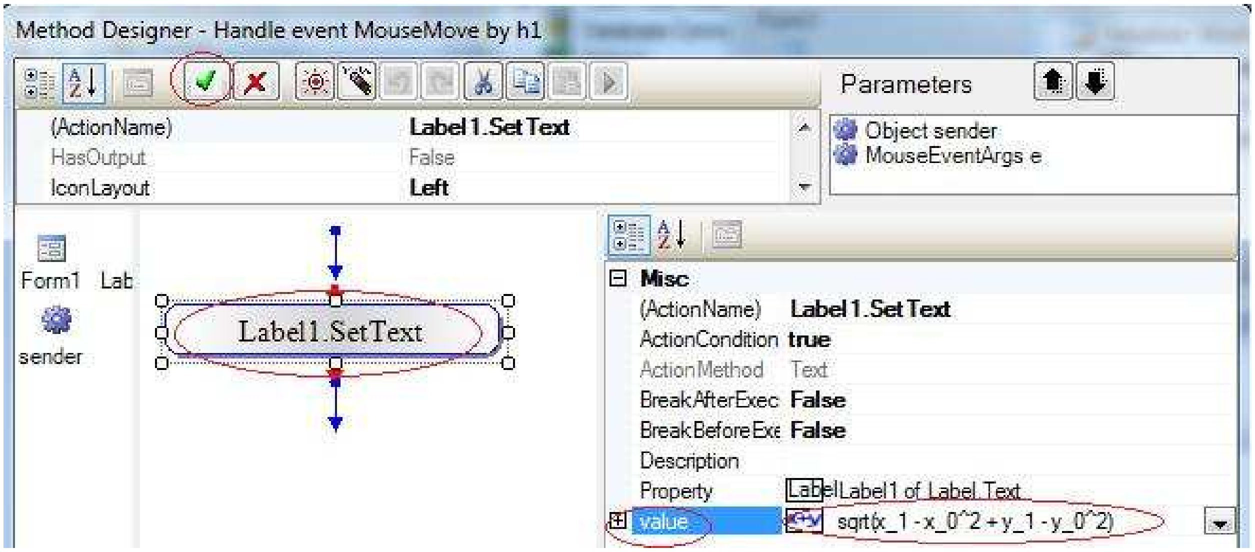


$$\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$$

Misc

x0	ClientSize.Width / 2
x1	e.X
y0	ClientSize.Height / 2
y1	e.Y

Click ✓ to finish creating this expression. The expression is used to provide value for parameter "value" of the action:



Method Designer - Handle event MouseMove by h1

(ActionName)	Label1.SetText
HasOutput	False
IconLayout	Left

Parameters

Object sender
MouseEventArgs e

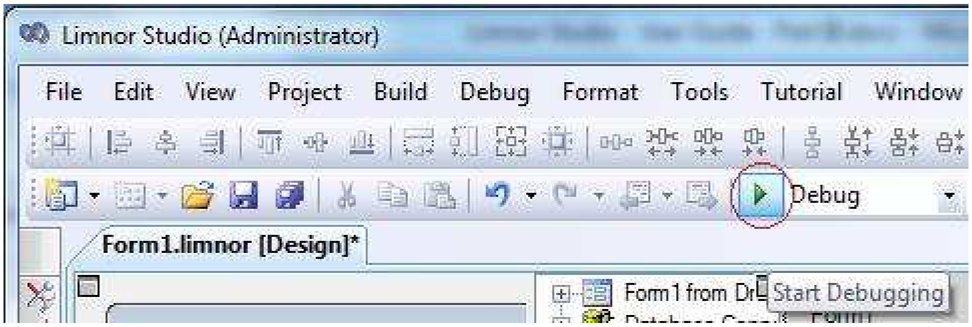
Misc

(ActionName)	Label1.SetText
ActionCondition	true
ActionMethod	Text
BreakAfterExec	False
BreakBeforeExe	False
Description	
Property	Label1 of Label.Text
value	sqrt(x_1 - x_0^2 + y_1 - y_0^2)

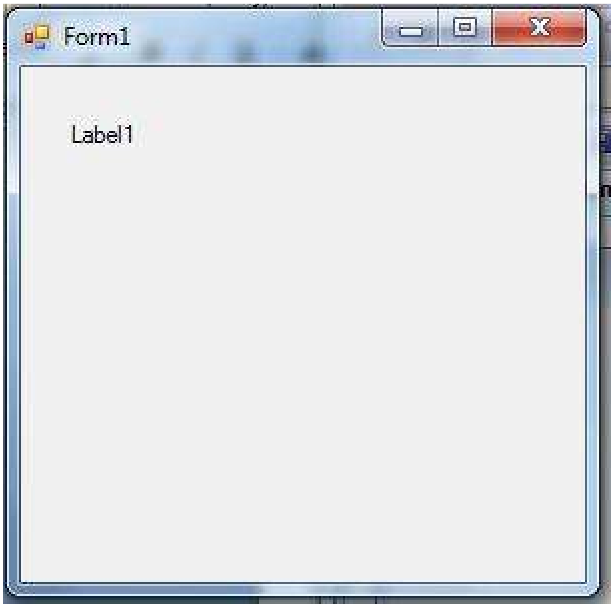
Click ✓ to finish modifying this event-handler method.

5.4 Test the sample

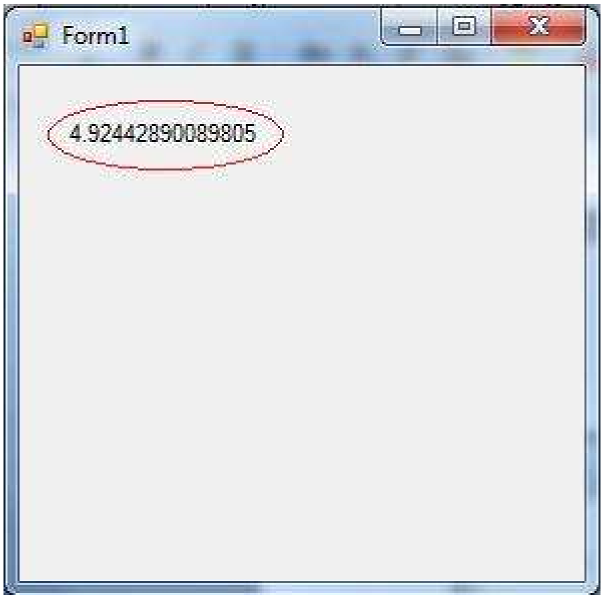
Click ▶ to compile and test the project:



The project is compiled and started. The first form appears:



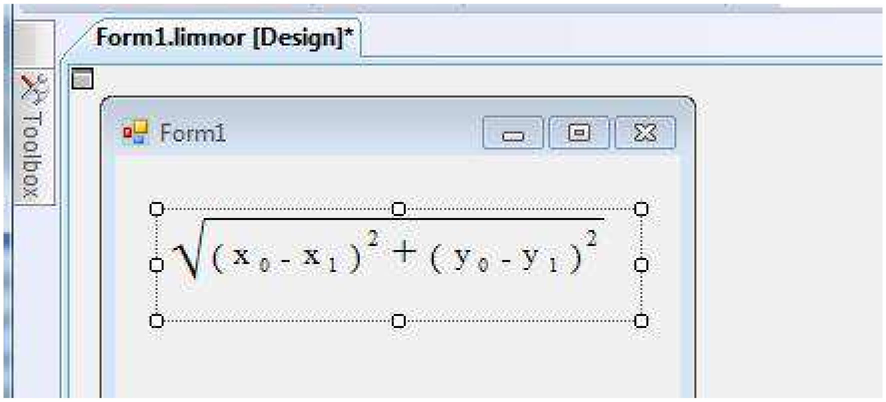
Now move mouse over the form. The distance between the mouse pointer and the center of the form appears in the Label. You can see that the value shown in the label getting smaller when moving mouse towards the center of the form; the value getting larger when moving the mouse away from the center of the form.



4.92442890089805

6 Use Math Expression Control

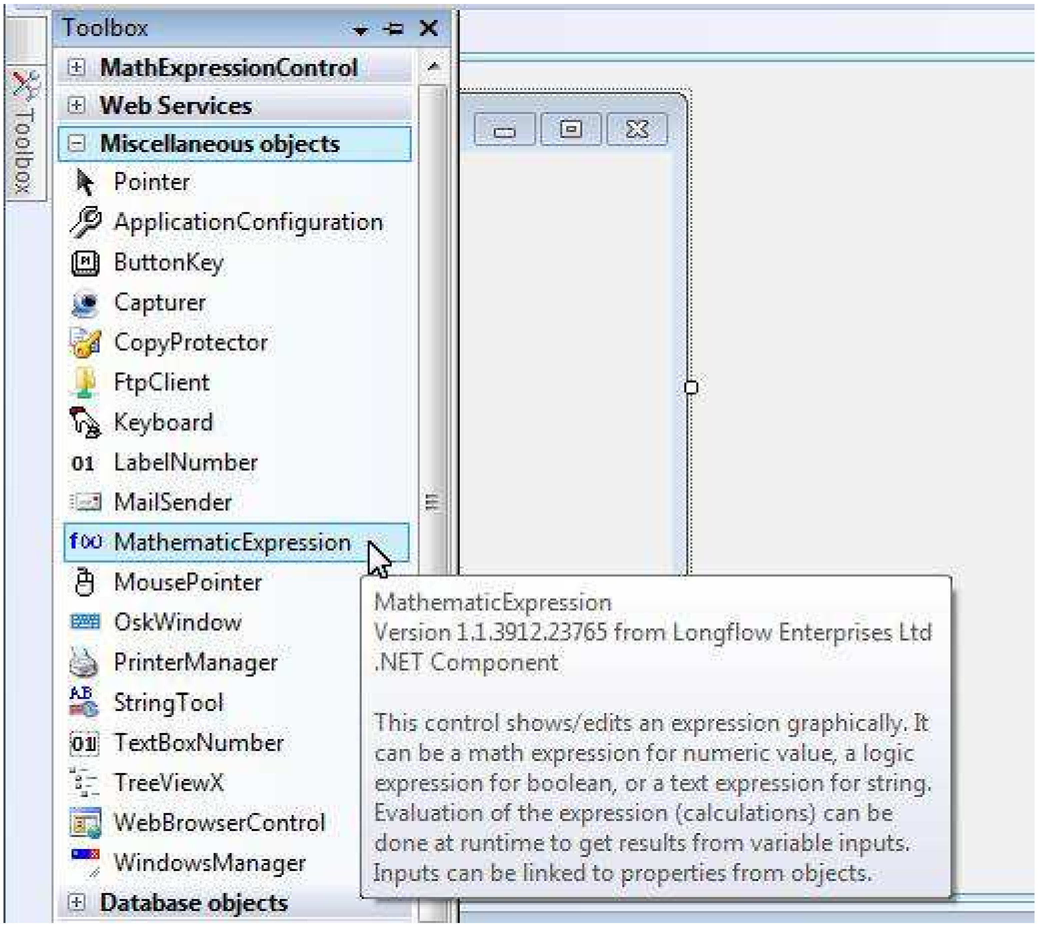
In previous samples all math expressions are used in action parameters. Math expressions may also be displayed as a user interface and used directly.



$$\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$$

6.1 Add Math Expression Control

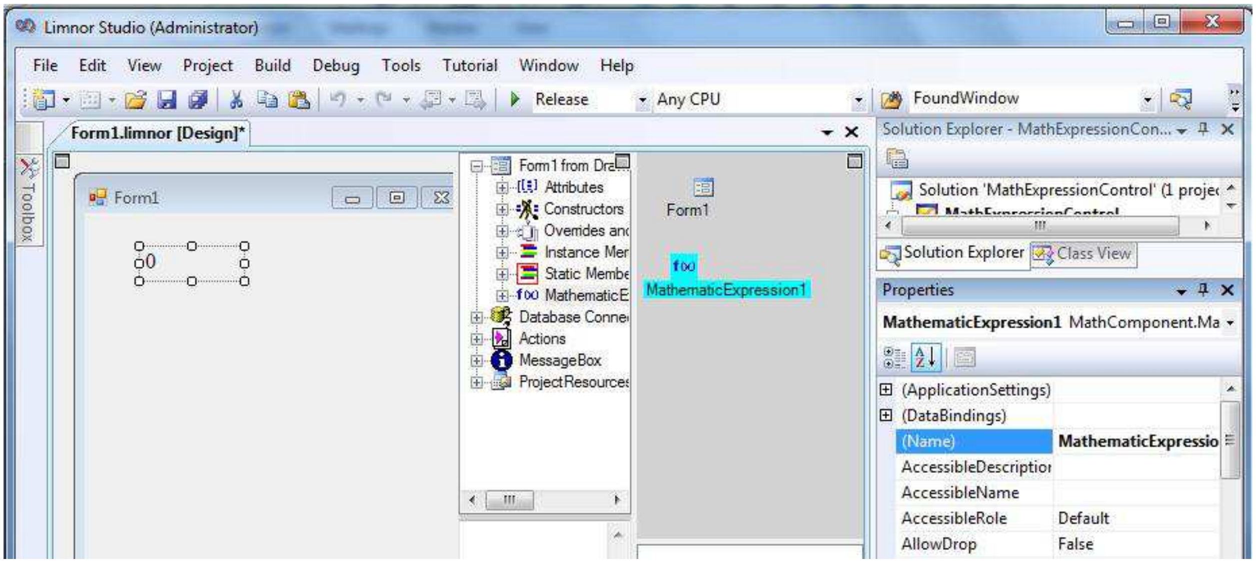
Drop a Math Expression control to a form to use it:



MathematicExpression
Version 1.1.3912.23765 from Longflow Enterprises Ltd
.NET Component

This control shows/edits an expression graphically. It can be a math expression for numeric value, a logic expression for boolean, or a text expression for string. Evaluation of the expression (calculations) can be done at runtime to get results from variable inputs. Inputs can be linked to properties from objects.

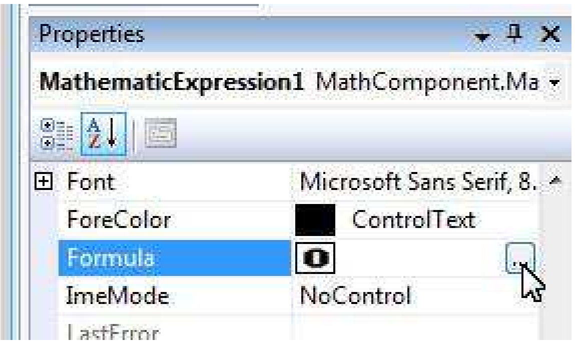
A Math Expression Control appears on the form:



(ApplicationSettings)	
(DataBindings)	
(Name)	MathematicExpressio
AccessibleDescriptior	
AccessibleName	
AccessibleRole	Default
AllowDrop	False

6.2 Enter formula at design time

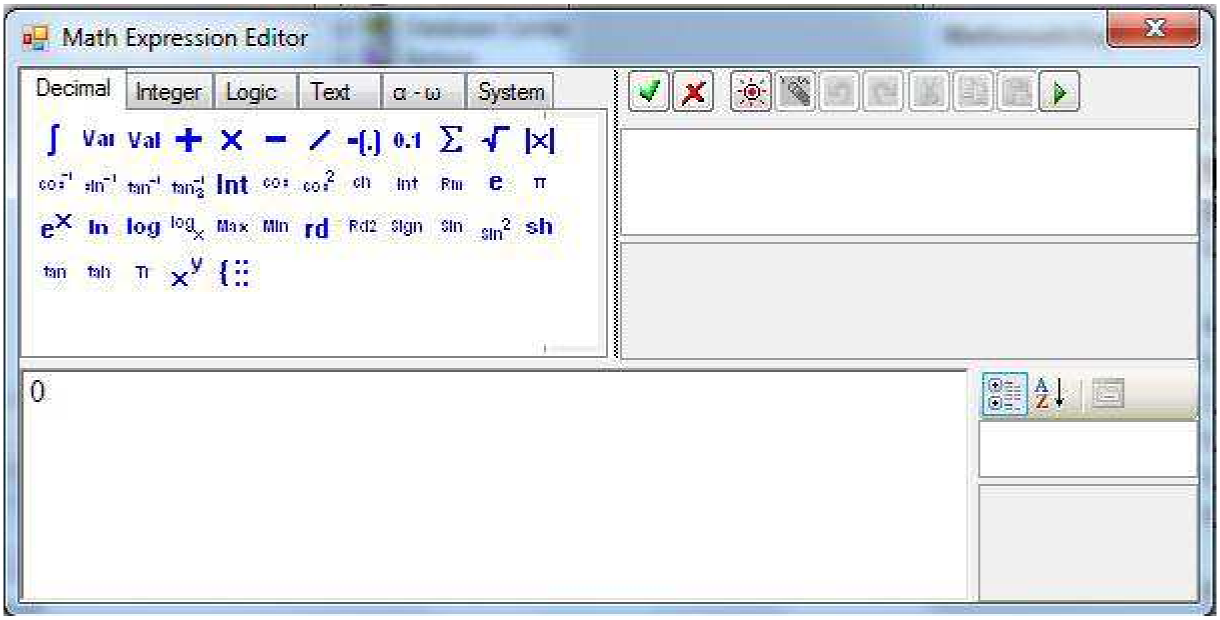
Set the Formula property at the design time:



MathematicExpression1 MathComponent.Ma

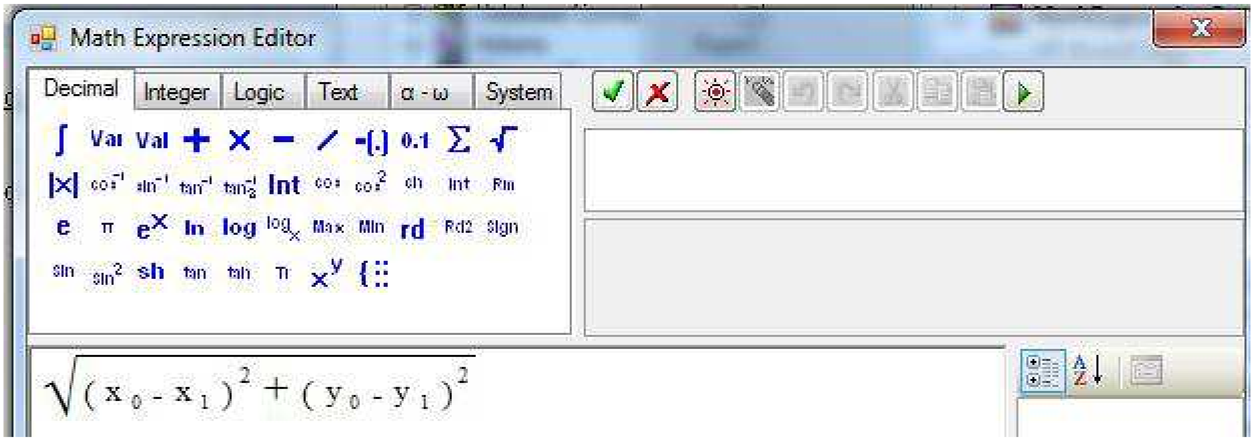
Font	Microsoft Sans Serif, 8.
ForeColor	ControlText
Formula	
ImeMode	NoControl
LastError	

The Math Expression Editor appears.



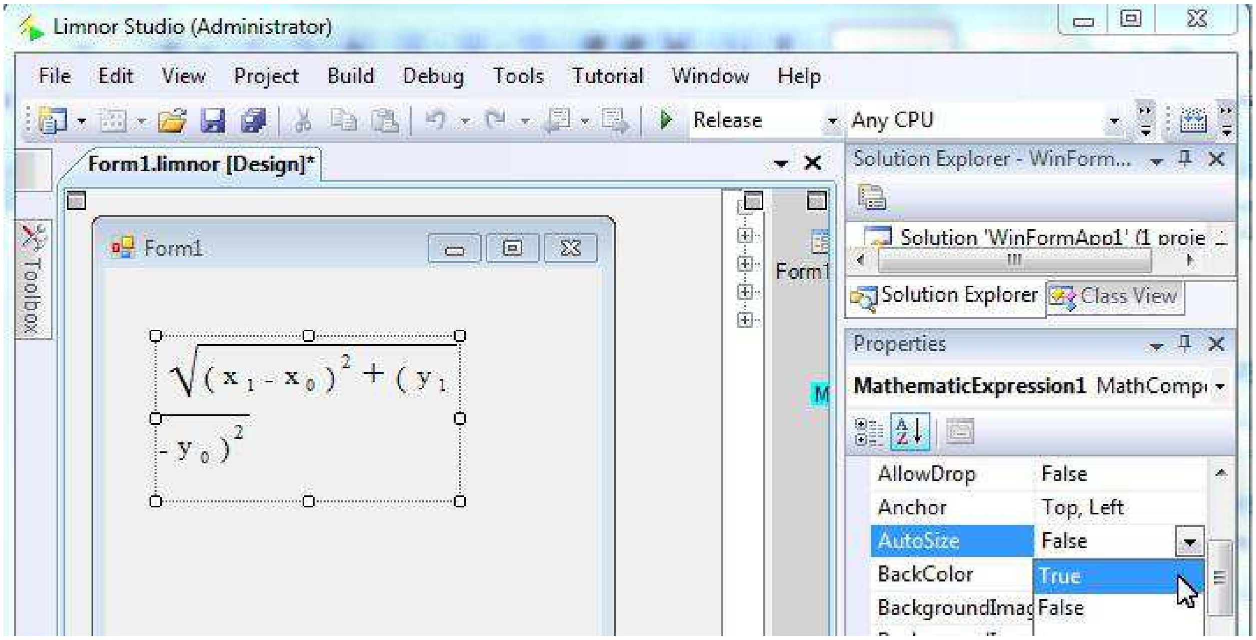
0

The use of the Math Expression Editor has been described previously. We do not need to go through the use of the Math Expression Editor. Suppose we entered the following formula:

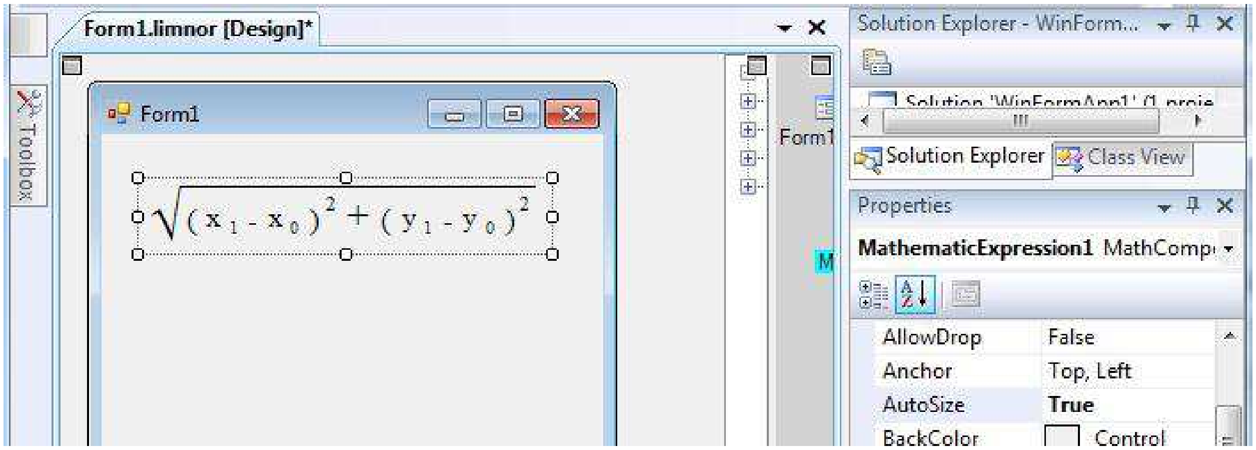


$$\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$$

The formula is displayed on the form. Set AutoSize property to True to resize it:


Limnor Studio (Administrator)
File Edit View Project Build Debug Tools Tutorial Window Help
Release
Any CPU
Form1.limnor [Design]*
Form1
Toolbox
$\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$
Solution Explorer - WinForm...
Solution 'WinFormApp1' (1 proje
Solution Explorer
Class View
Properties
MathematicExpression1 MathComp
M

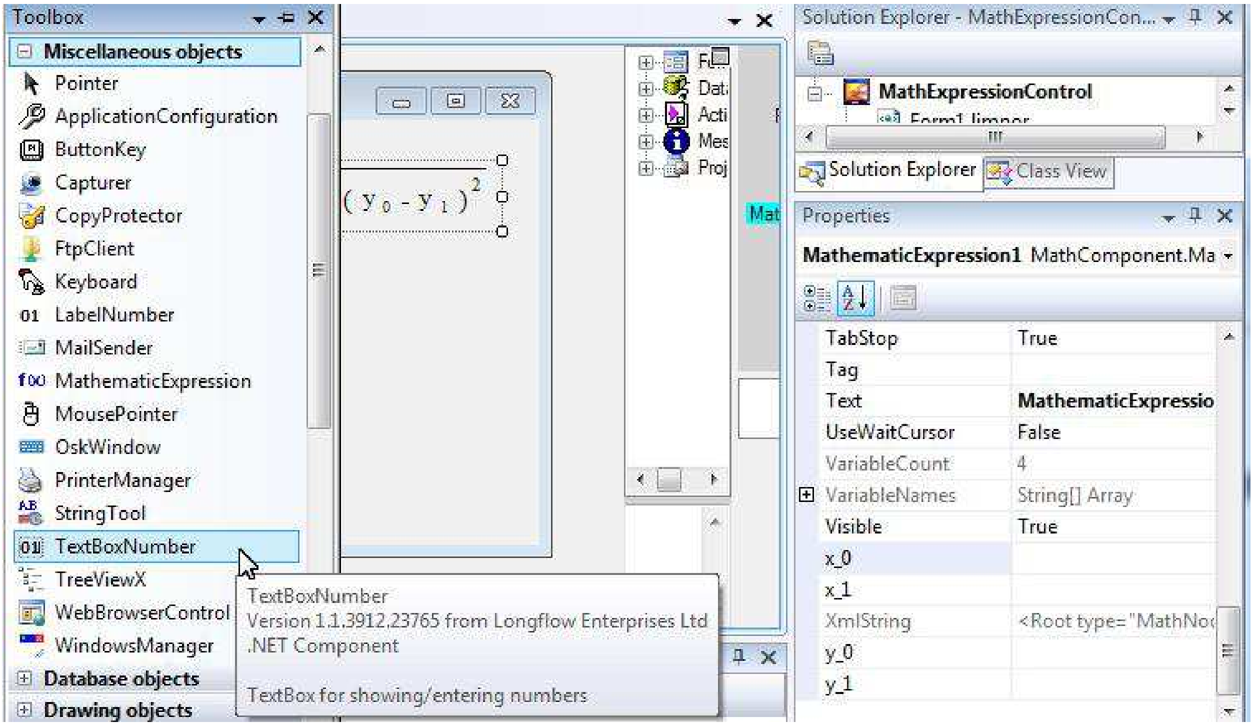
AllowDrop False
Anchor Top, Left
AutoSize False
BackColor
BackgroundImag
True
False


Form1.limnor [Design]*
Form1
Toolbox
$\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$
Solution Explorer - WinForm...
Solution 'WinFormApp1' (1 proje
Solution Explorer
Class View
Properties
MathematicExpression1 MathComp
M

AllowDrop False
Anchor Top, Left
AutoSize True
BackColor Control

6.3 Map variables to properties at design time

The variables in the formula become the properties of the Math Expression Control. We may link the variables to properties in the program to provide data for calculating the formula value. Let's add some text boxes for providing data for this formula:



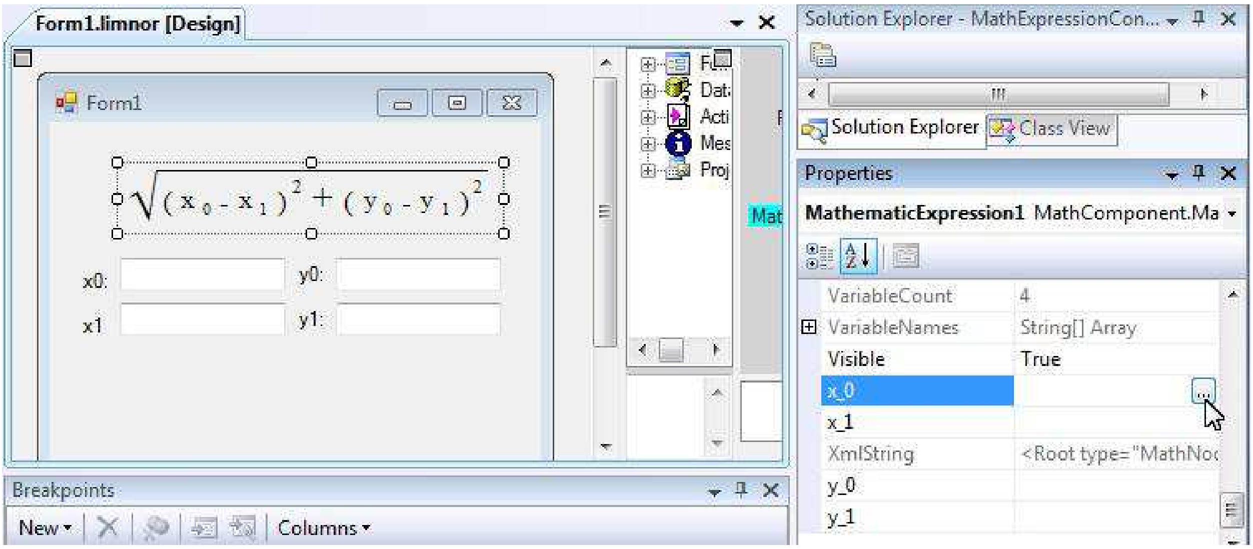
TextBoxNumber
Version 1.1.3912.23765 from Longflow Enterprises Ltd
.NET Component

TextBox for showing/entering numbers

MathematicExpression1 MathComponent.Ma

TabStop	True
Tag	
Text	MathematicExpressio
UseWaitCursor	False
VariableCount	4
VariableNames	String[] Array
Visible	True
x_0	
x_1	
XmlString	<Root type="MathNo
y_0	
y_1	

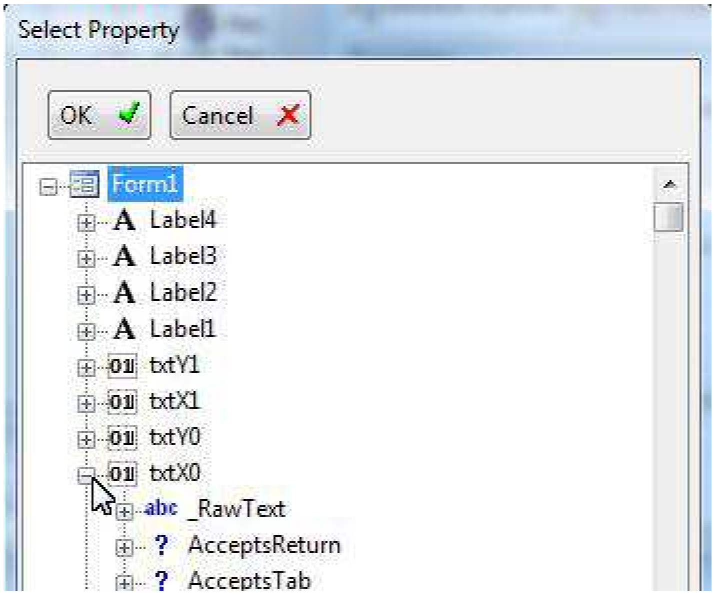
Link x0 to a text box:



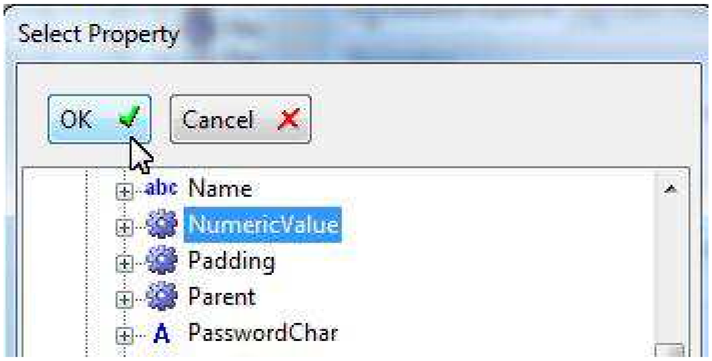
MathematicExpression1 MathComponent.Ma

VariableCount	4
VariableNames	String[] Array
Visible	True
x_0	...
x_1	
XmlString	<Root type="MathNo
y_0	
y_1	

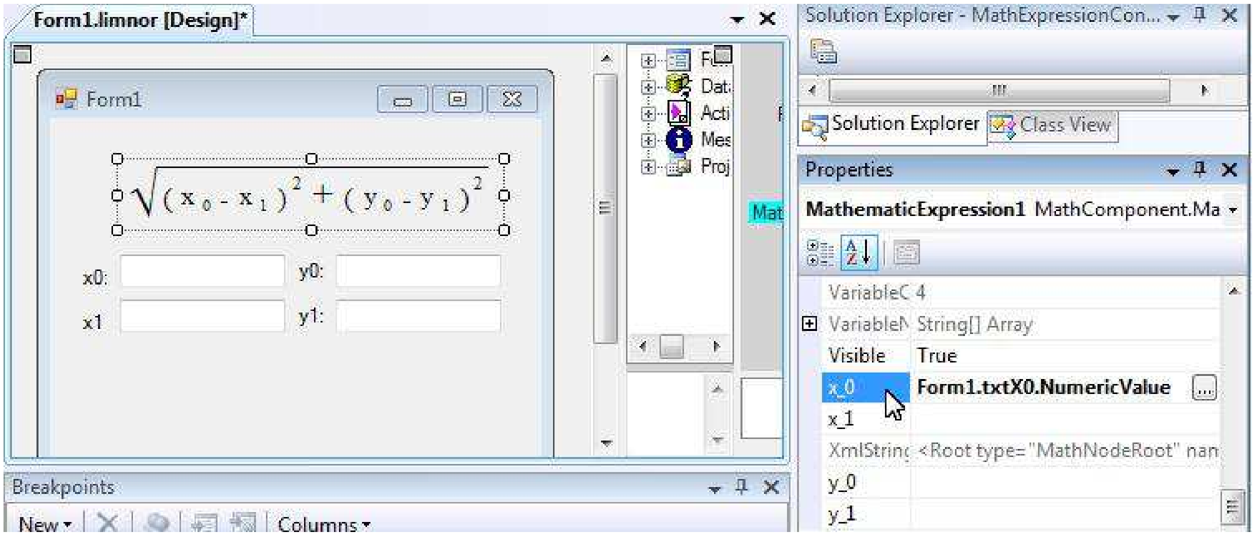
Expand the text box txtX0:



Select NumericValue property:



We linked variable x0 to the text box txtX0:

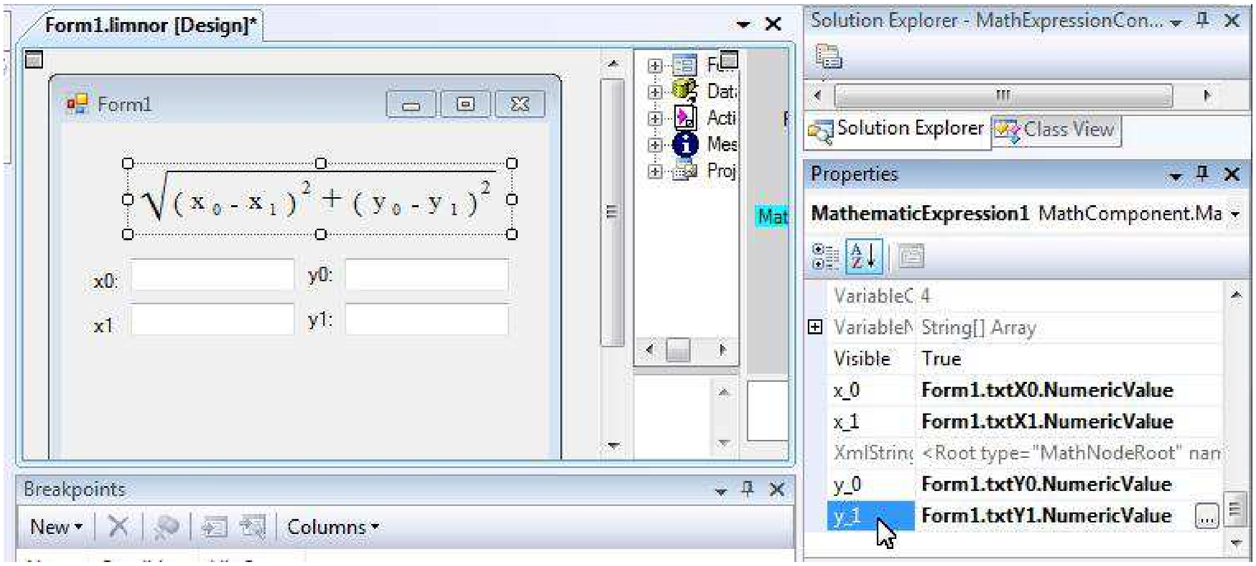


$$\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$$

MathematicExpression1 MathComponent.Ma

VariableC	4
VariableN	String[] Array
Visible	True
x_0	Form1.txtX0.NumericValue
x_1	
XmlString	<Root type="MathNodeRoot" nan
y_0	
y_1	

We may make links for the other variables:

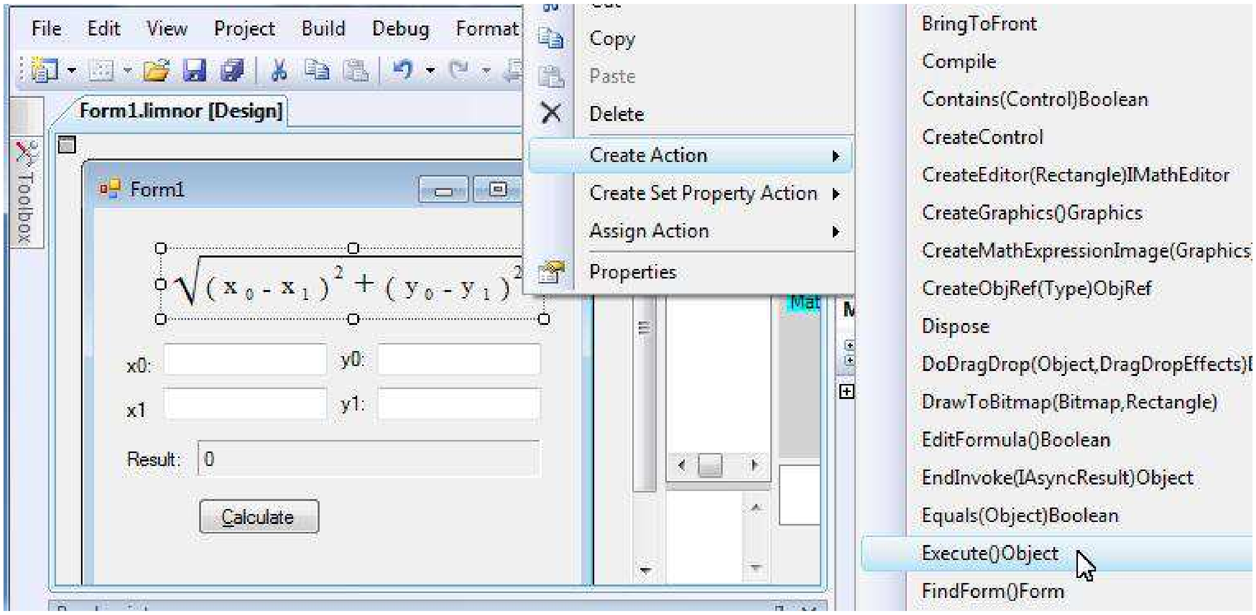

Form1.limnor [Design]*
Form1
$\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$
x0:
y0:
x1
y1:
Breakpoints
New
Columns
Solution Explorer - MathExpressionCon...
Solution Explorer
Class View
Properties
MathematicExpression1 MathComponent.Ma

VariableC	4
VariableN	String[] Array
Visible	True
x_0	Form1.txtX0.NumericValue
x_1	Form1.txtX1.NumericValue
XmlStrin	<Root type="MathNodeRoot" nan
y_0	Form1.txtY0.NumericValue
y_1	Form1.txtY1.NumericValue

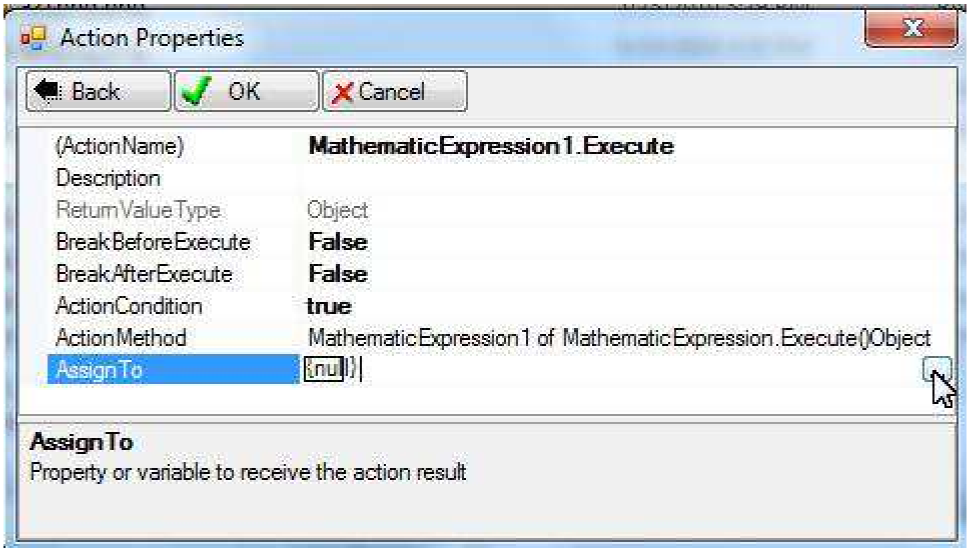
Let's use a label to display calculation result. To do the calculation, create an Execute action.

6.4 Use Execute action to do calculation

Right-click the Math Expression Control; choose "Create Action". Choose "Execute" method:



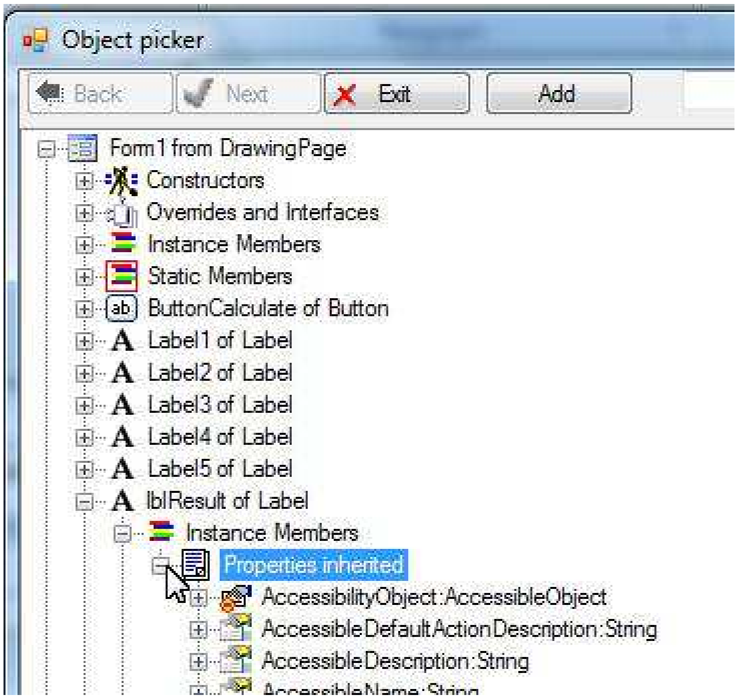
Set the AssignTo property of the action:



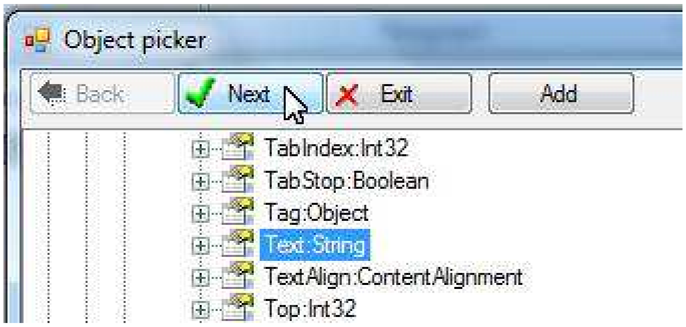
(ActionName)	**MathematicExpression1.Execute**
Description	
ReturnValueType	Object
BreakBeforeExecute	**False**
BreakAfterExecute	**False**
ActionCondition	**true**
ActionMethod	MathematicExpression1 of MathematicExpression.Execute()Object
AssignTo	{null}

AssignTo
Property or variable to receive the action result

Select the Text property of the label:

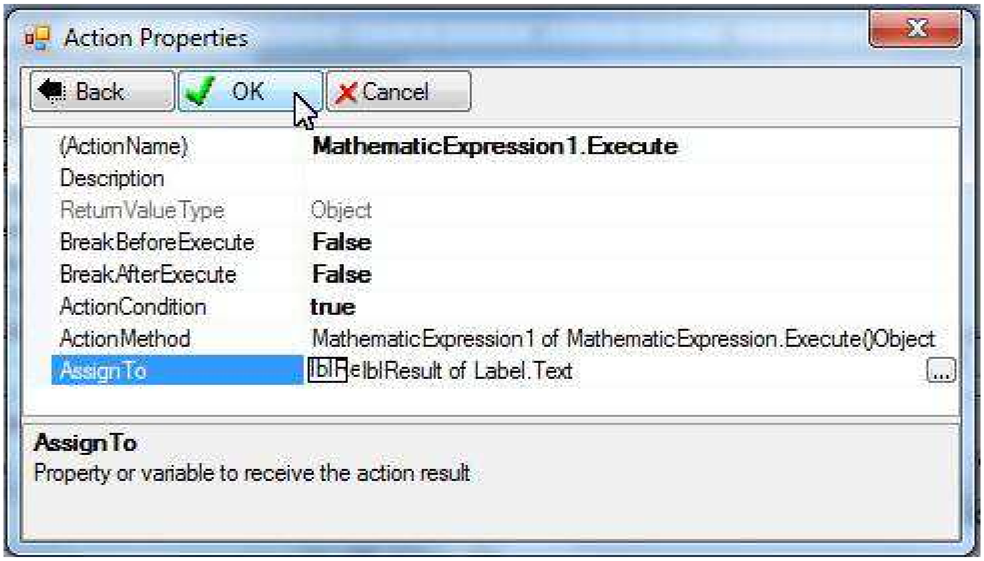


Form1 from DrawingPage
Constructors
Overrides and Interfaces
Instance Members
Static Members
ButtonCalculate of Button
Label1 of Label
Label2 of Label
Label3 of Label
Label4 of Label
Label5 of Label
lblResult of Label
Instance Members
Properties inherited
AccessibilityObject:AccessibleObject
AccessibleDefaultActionDescription:String
AccessibleDescription:String
AccessibleName:String



TabIndex:Int32
TabStop:Boolean
Tag:Object
Text:String
TextAlign:ContentAlignment
Top:Int32

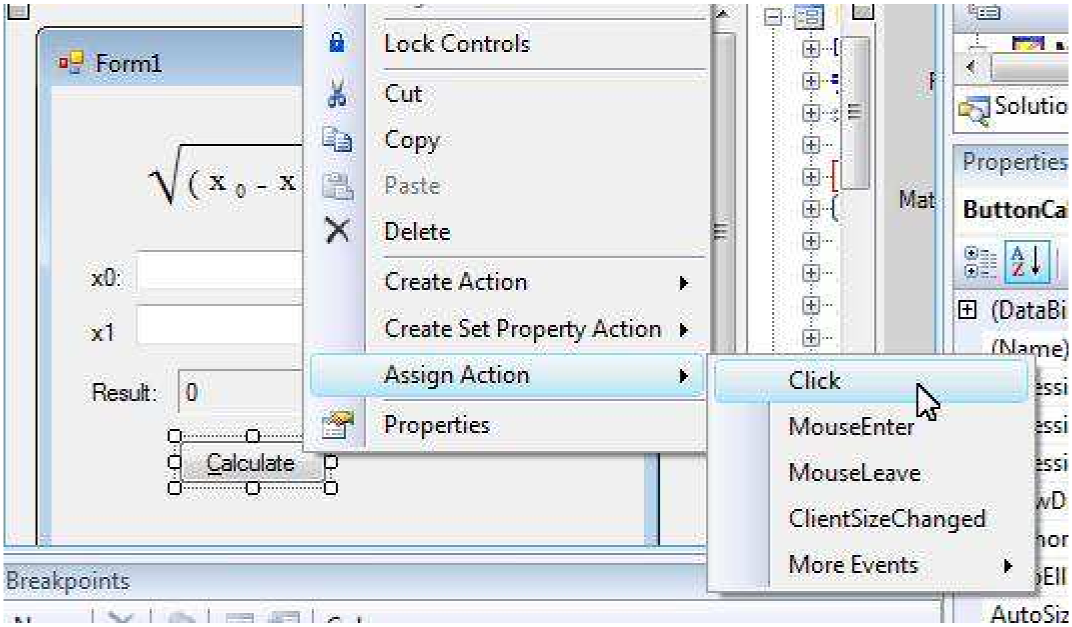
Click OK to finish creating the action:

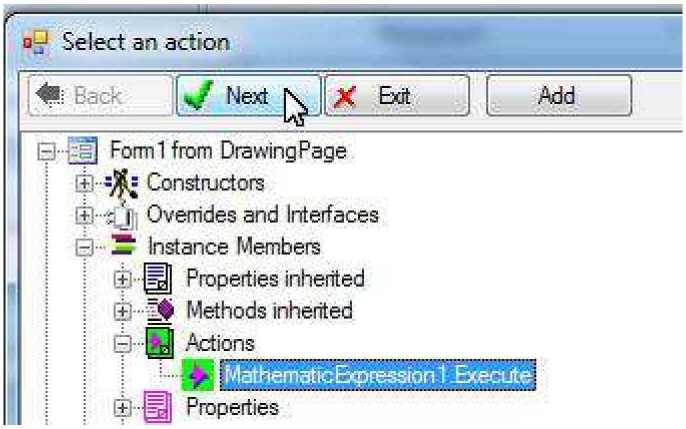


(ActionName)	**MathematicExpression1.Execute**
Description	
ReturnValueType	Object
BreakBeforeExecute	**False**
BreakAfterExecute	**False**
ActionCondition	**true**
ActionMethod	MathematicExpression1 of MathematicExpression.Execute()Object
AssignTo	lblResult of Label.Text

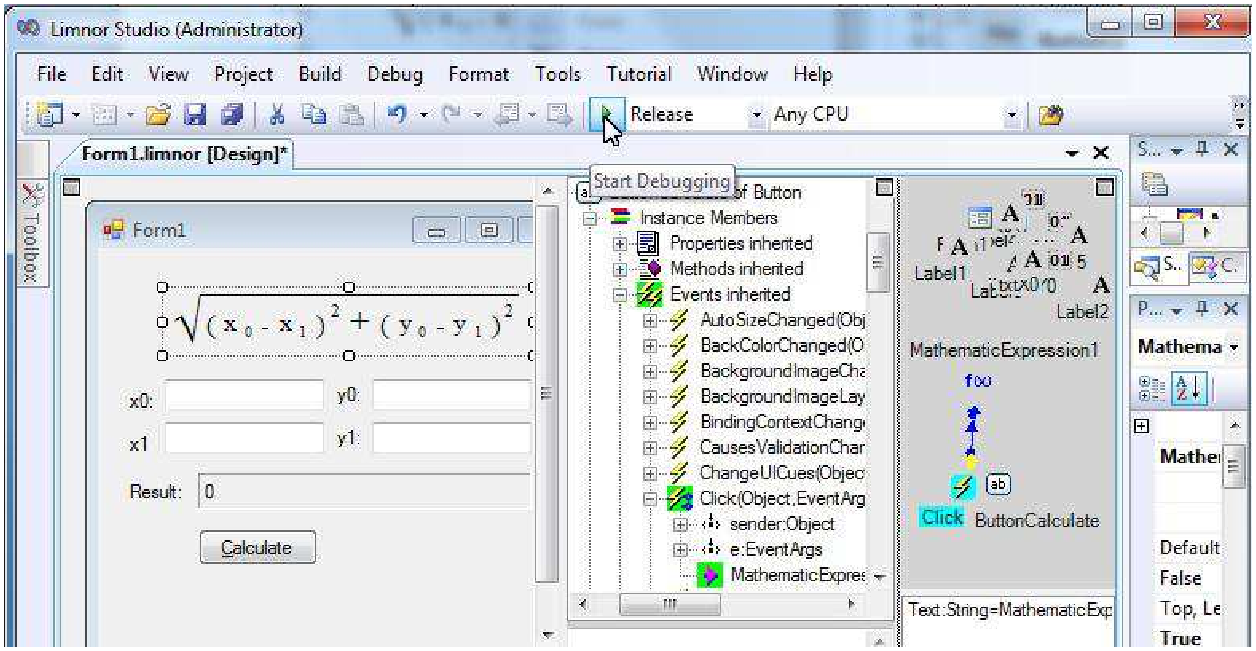
AssignTo
Property or variable to receive the action result

Assign the action to the Click event of the button:

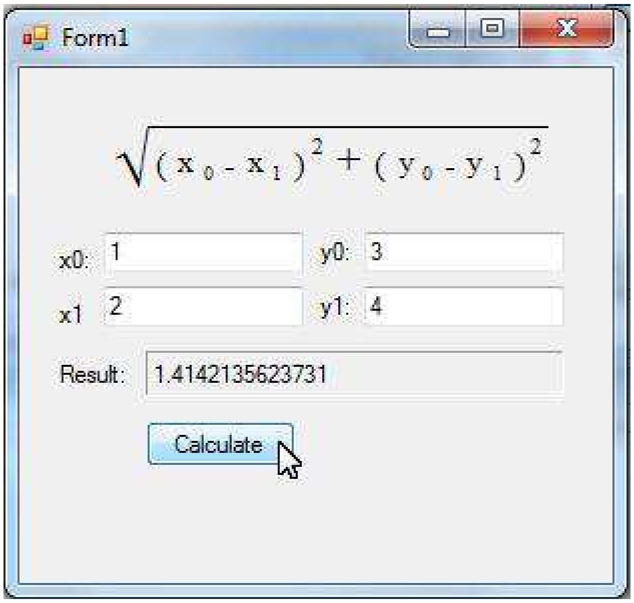




We may test it now:



Type in some values in the text boxes. Click the button. The calculation result is displayed in the label:



x0: 1 y0: 3

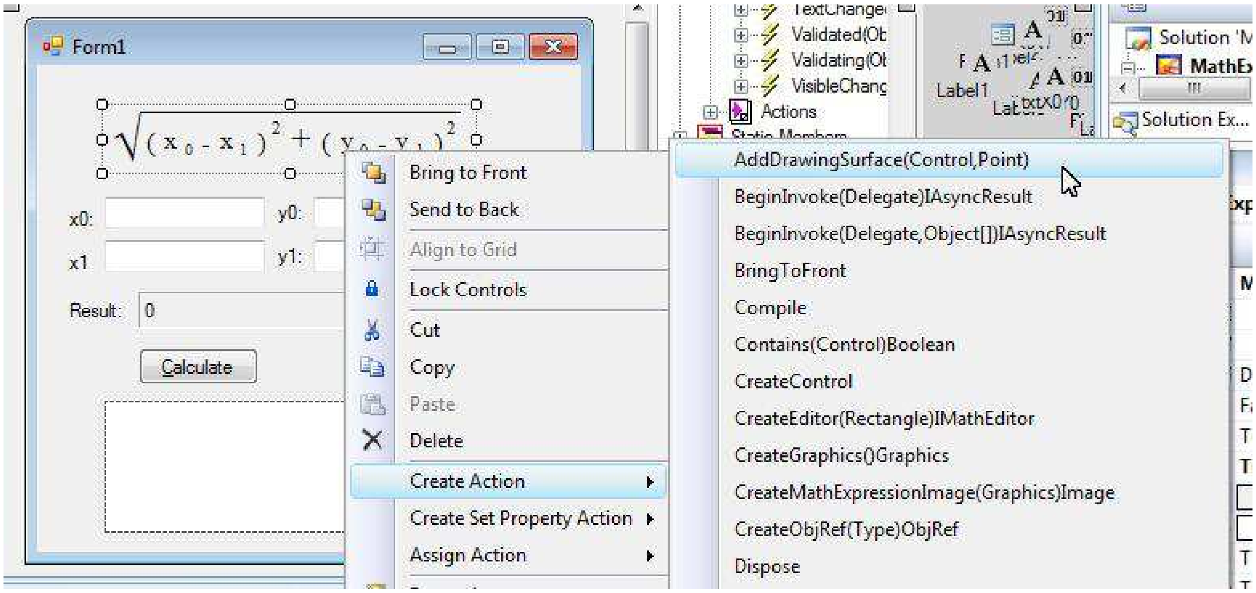
x1 2 y1: 4

Result: 1.4142135623731

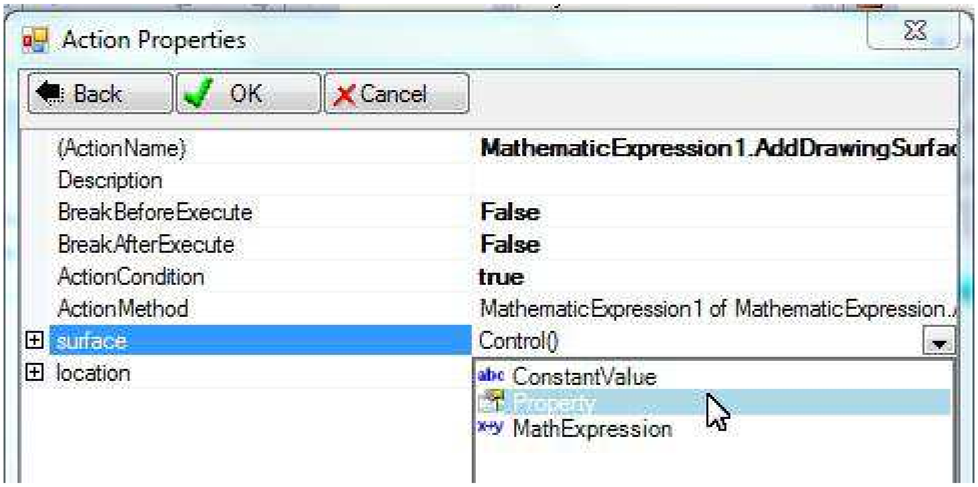
6.5 Display Math Expression on Controls

A Math Expression can be displayed on any controls, and on more than one controls. Here we use a Picture Box to demonstrate this feature. It works for other types of controls for examples, labels, text boxes, buttons, etc.

Right-click the Math Expression Control; choose “Create Action”; choose “AddDrawingSurface”:

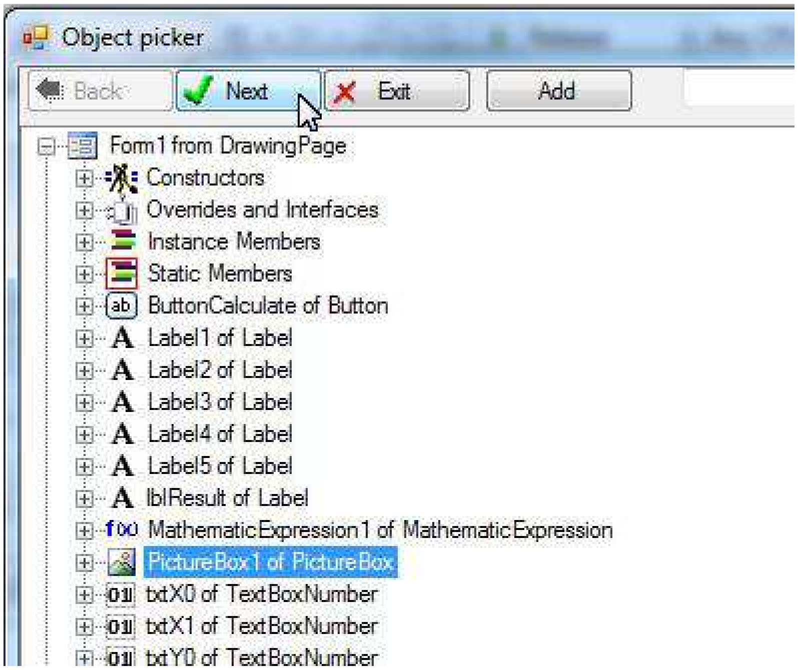


Choose Property for the surface:

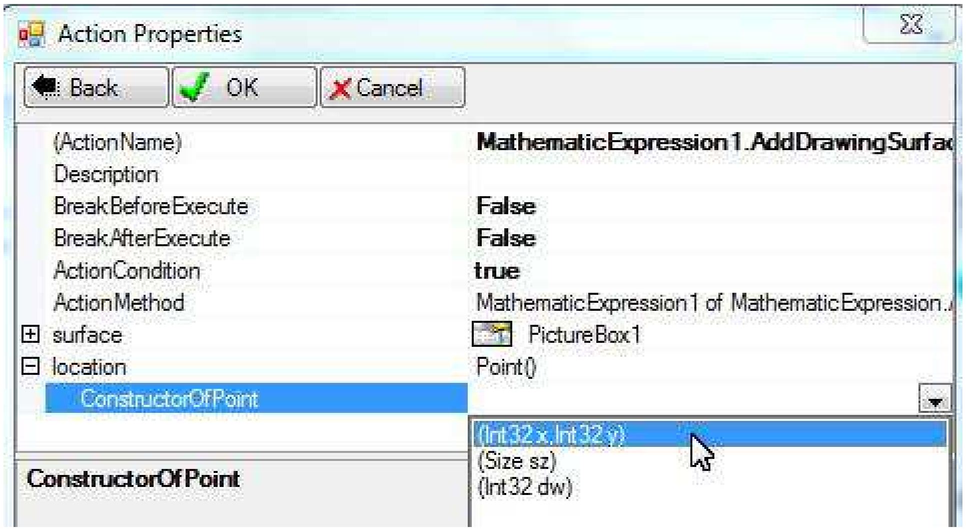


(ActionName)	MathematicExpression1.AddDrawingSurfac
Description	
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	MathematicExpression1 of MathematicExpression./
surface	Control()
location	

Select the Picture Box:



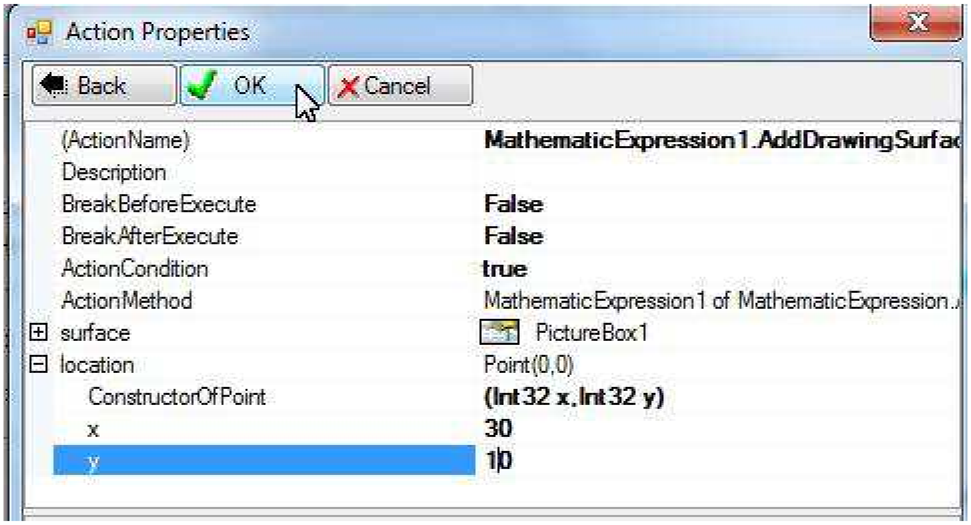
Enter starting point for displaying the Math Expression:



(ActionName)	MathematicExpression1.AddDrawingSurfac
Description	
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	MathematicExpression1 of MathematicExpression./
⊞ surface	PictureBox1
⊟ location	Point()
ConstructorOfPoint	

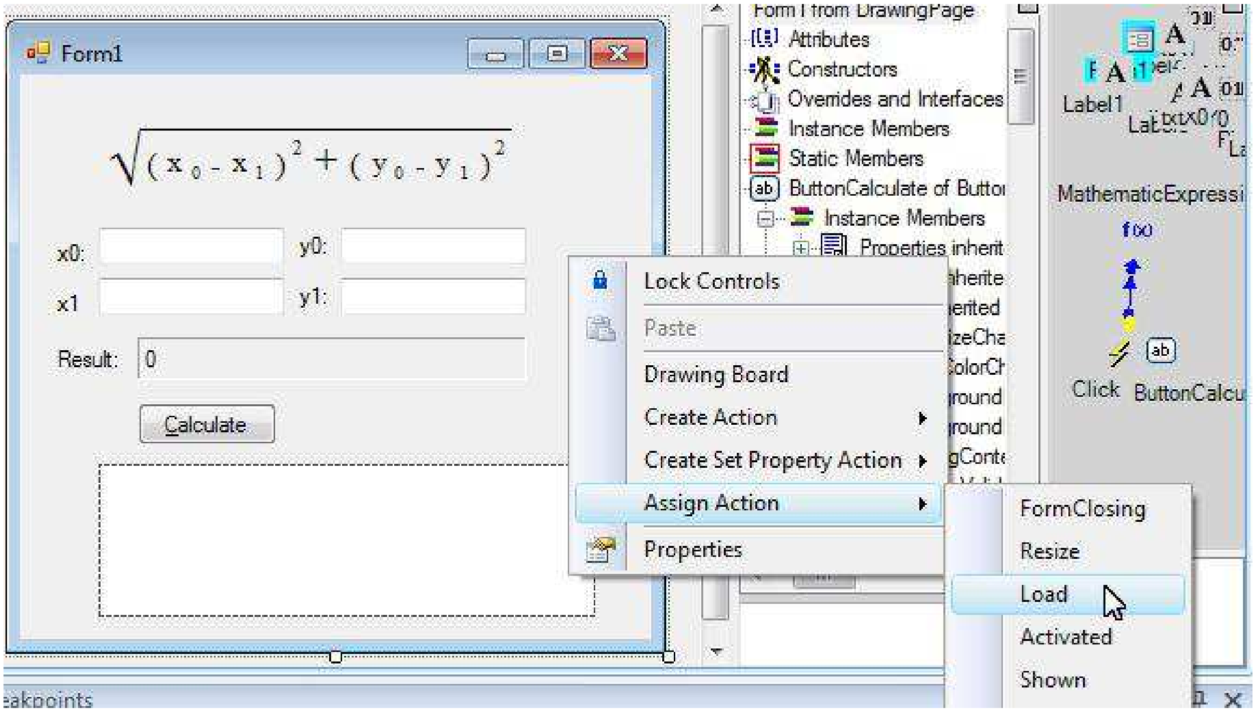
(Int32 x,Int32 y)
(Size sz)
(Int32 dw)

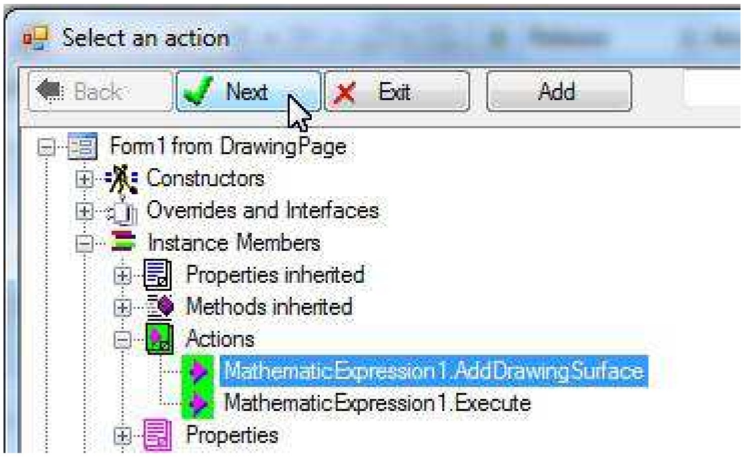
ConstructorOfPoint



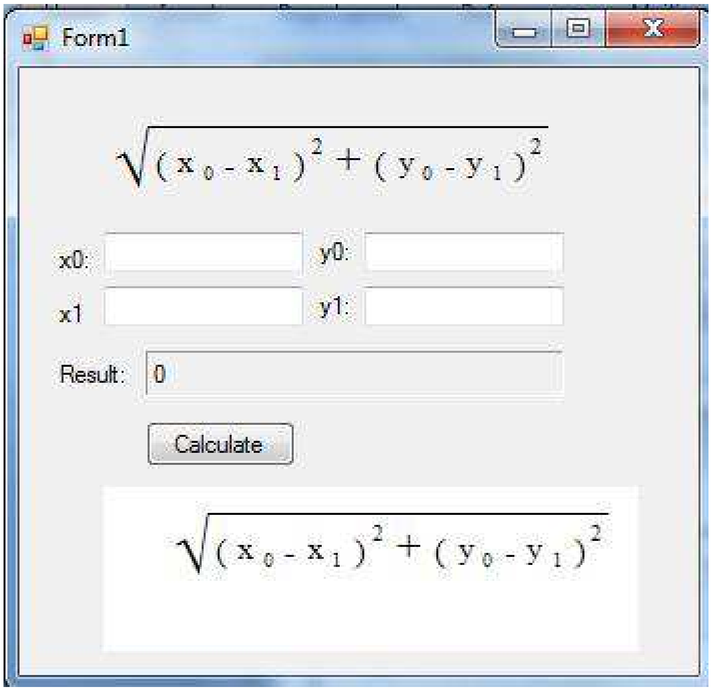
(ActionName)	MathematicExpression1.AddDrawingSurfac
Description	
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	MathematicExpression1 of MathematicExpression./
⊞ surface	PictureBox1
⊟ location	Point(0,0)
ConstructorOfPoint	(Int32 x,Int32 y)
x	30
y	10

Let’s assign this action to the Load event of the Form:





Run this application. We can see that the Math Expression is displayed on the Picture Box:

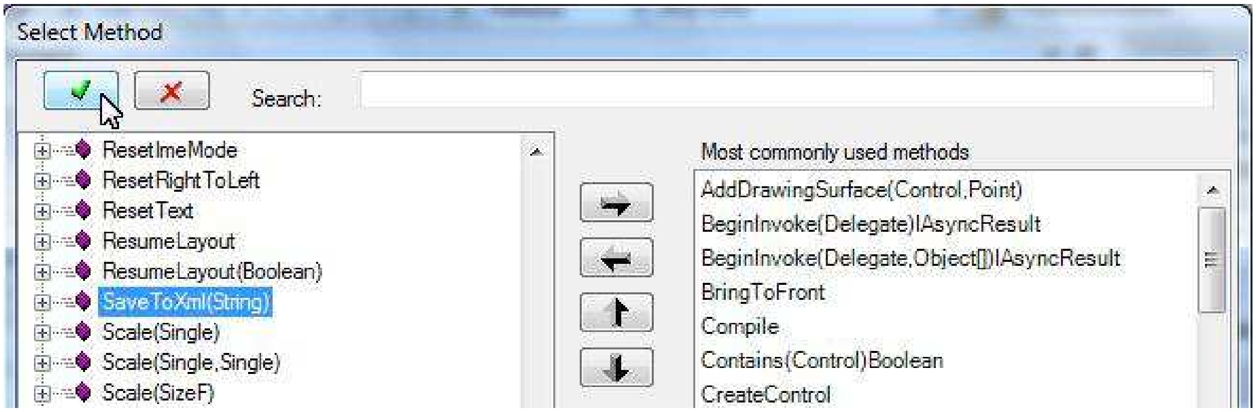

Form1
$\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$
x0:
y0:
x1
y1:
Result:
0
Calculate
$\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$

To remove the display of the Math Expression from a control surface, use a RemoveDrawingSurface action.

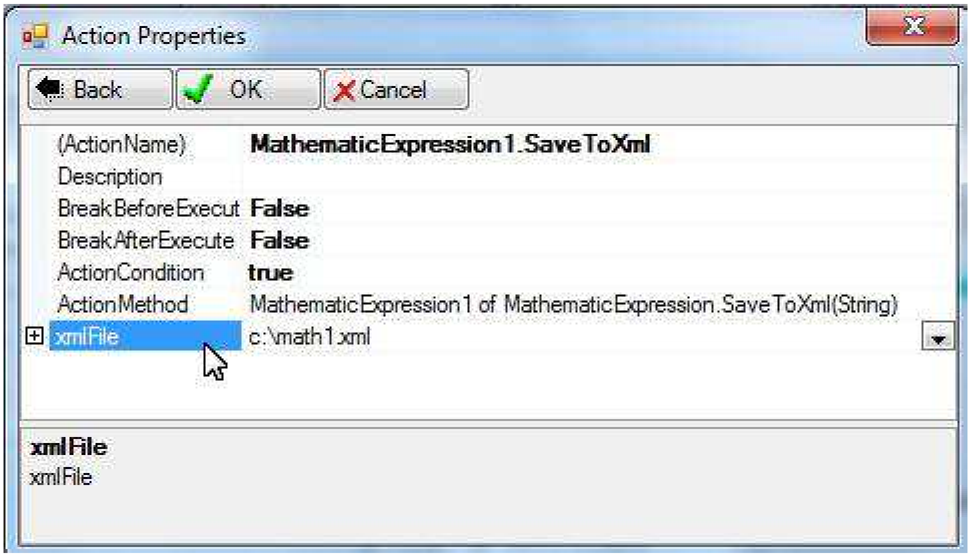
6.6 Save Formula to an XML File at Runtime

The formula in a Math Expression is represented in XML text, which can be retrieved by the XmlString property. A method, SaveToXml, is provided to save the XmlString property to a file.

Right-click the Math Expression Control; choose "Create Action". Choose "More Methods" and "All Methods". Select "SaveToXml":



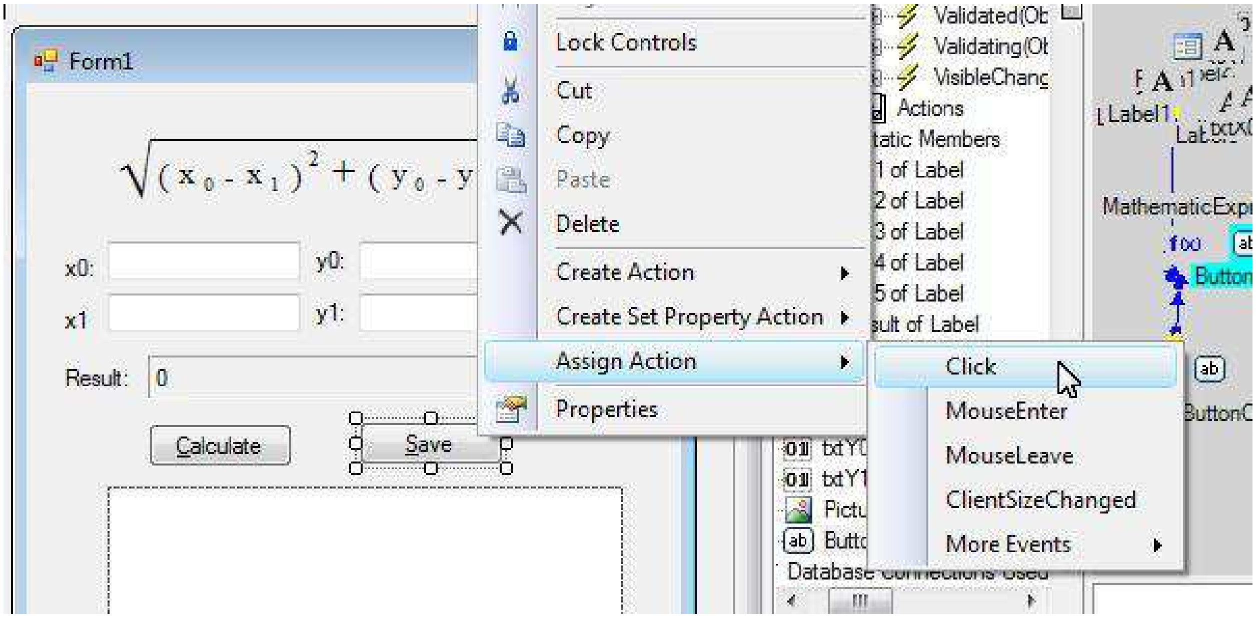
For simplicity, give it a constant file name:

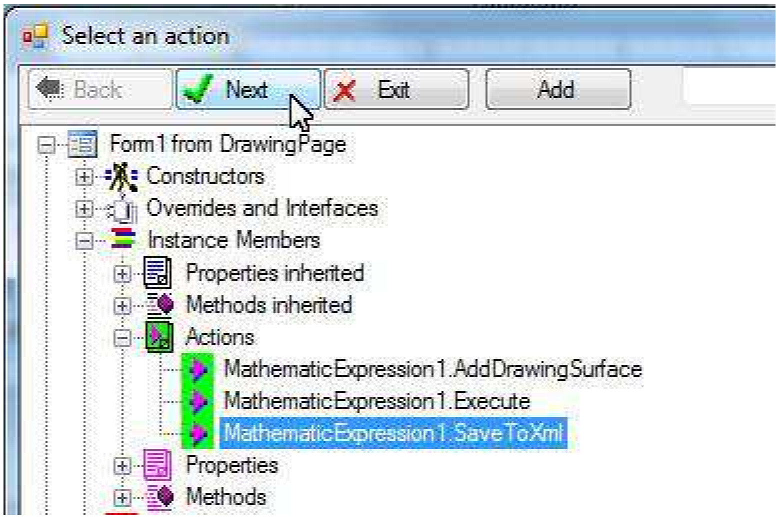


(ActionName)	**MathematicExpression1.SaveToXml**
Description	
BreakBeforeExecut	**False**
BreakAfterExecute	**False**
ActionCondition	**true**
ActionMethod	MathematicExpression1 of MathematicExpression.SaveToXml(String)
xmlFile	c:\math1.xml

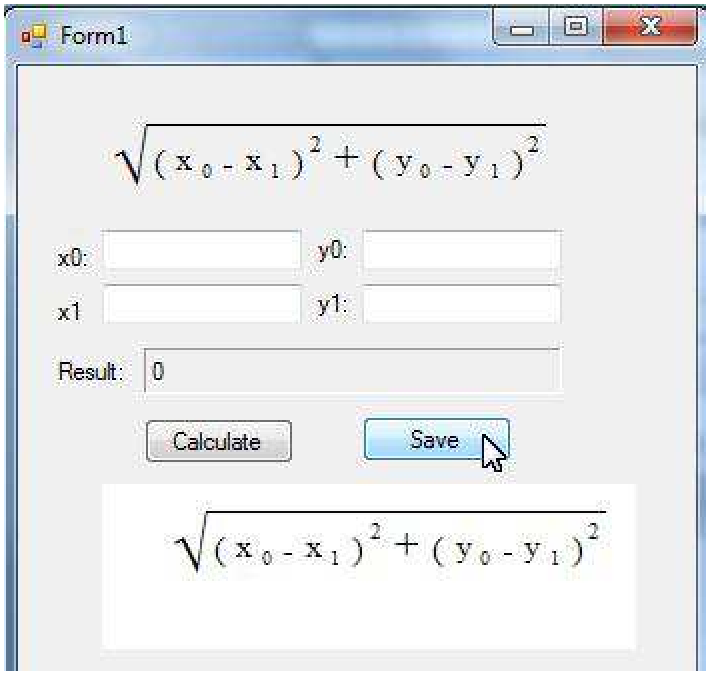
xmlFile
xmlFile

Assign the action to the Click event of a button:

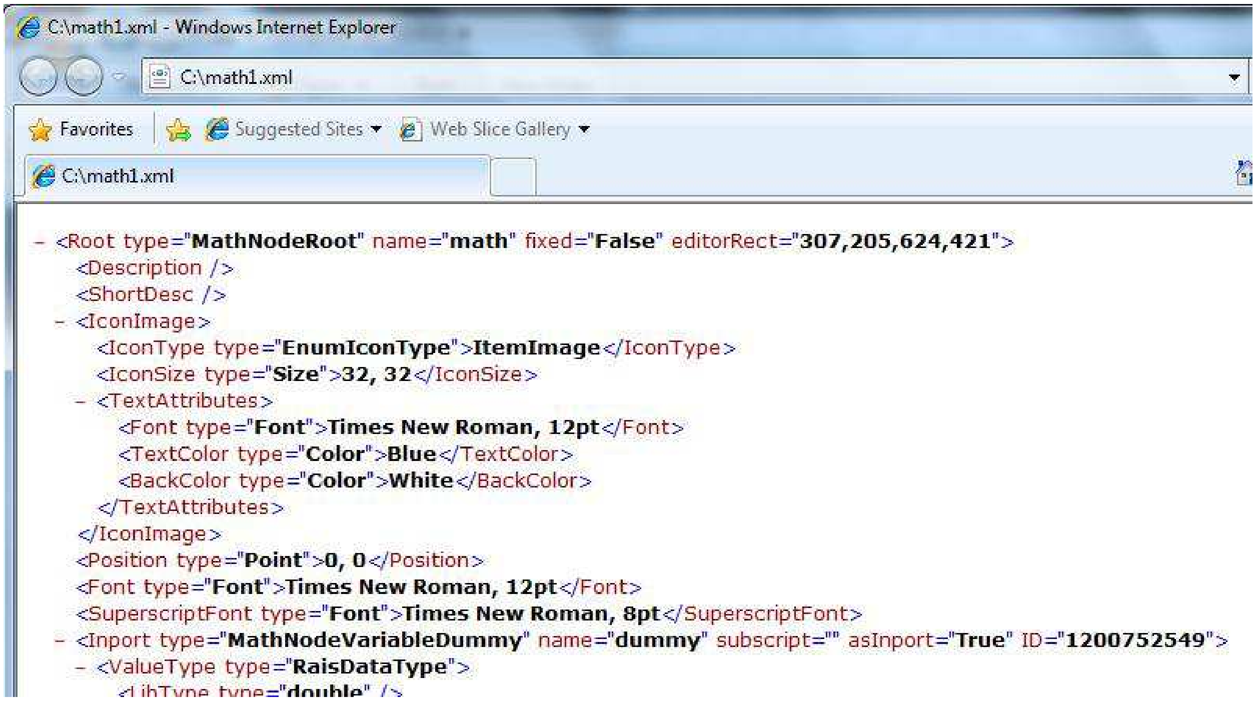




Run the application. Click the Save button.


Form1
$\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$
x0:
y0:
x1
y1:
Result:
0
Calculate
Save
$\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$

An XML file is generated:

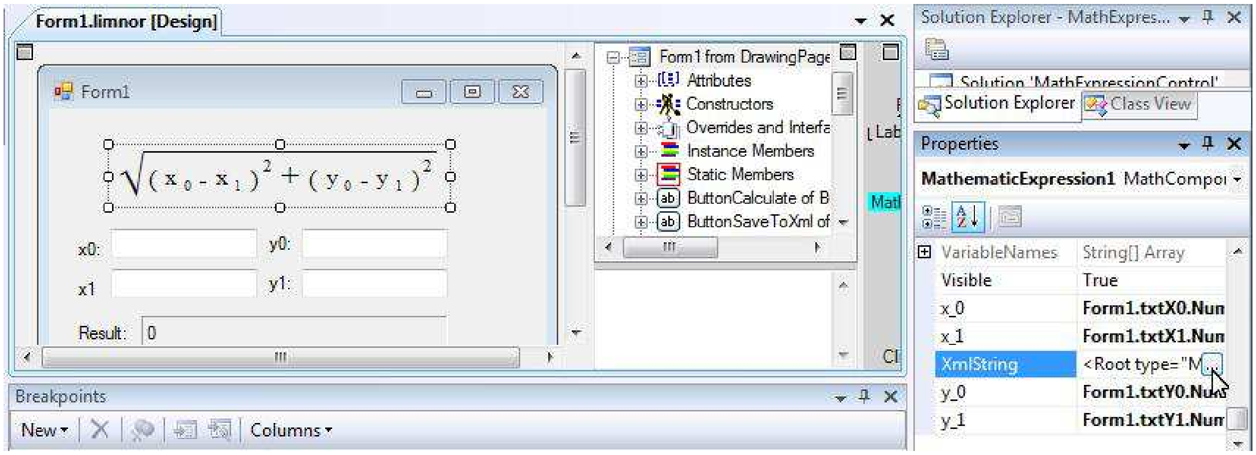


```
- <Root type="MathNodeRoot" name="math" fixed="False" editorRect="307,205,624,421">
    <Description />
    <ShortDesc />
  - <IconImage>
      <IconType type="EnumIconType">ItemImage</IconType>
      <IconSize type="Size">32, 32</IconSize>
    - <TextAttributes>
        <Font type="Font">Times New Roman, 12pt</Font>
        <TextColor type="Color">Blue</TextColor>
        <BackColor type="Color">White</BackColor>
      </TextAttributes>
    </IconImage>
    <Position type="Point">0, 0</Position>
    <Font type="Font">Times New Roman, 12pt</Font>
    <SuperscriptFont type="Font">Times New Roman, 8pt</SuperscriptFont>
  - <Inport type="MathNodeVariableDummy" name="dummy" subscript="" asInport="True" ID="1200752549">
    - <ValueType type="RaisDataType">
        <LibType type="double" />
```

6.7 Save and Load Formula Xml at Design Time

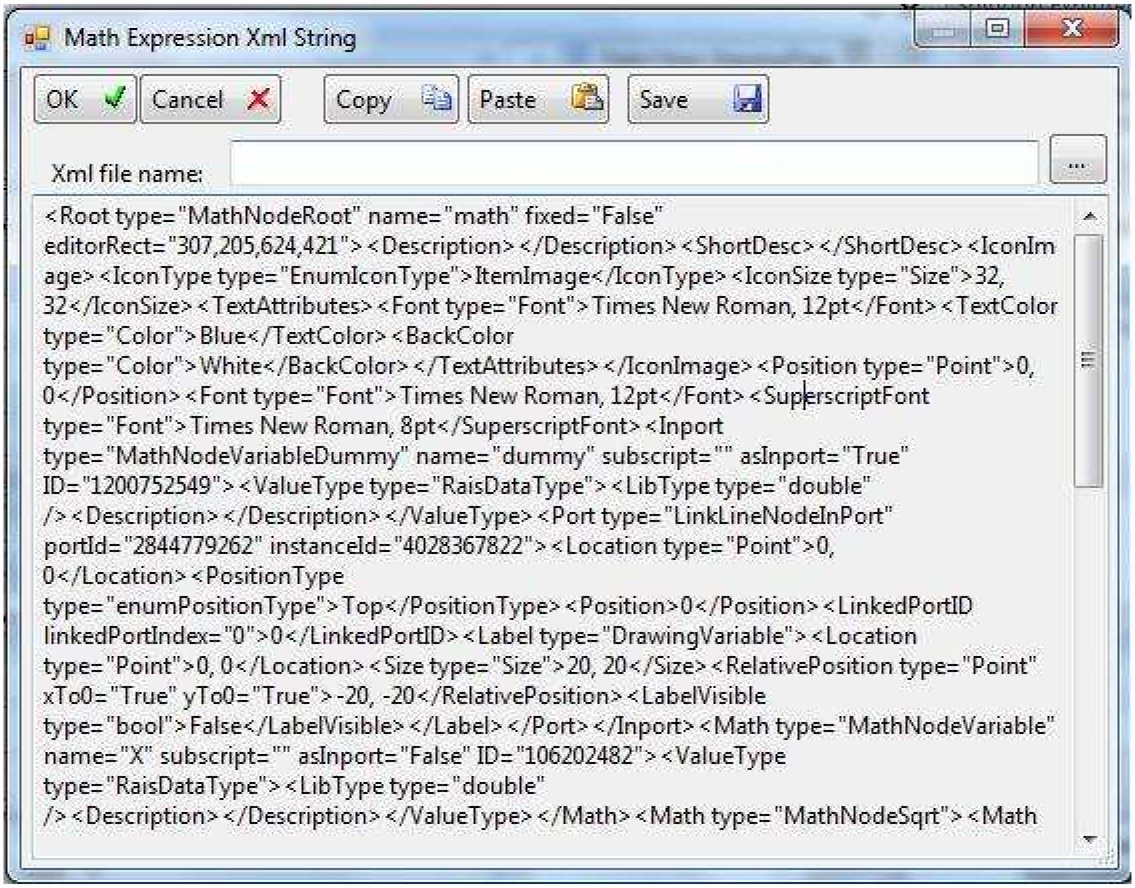
XmlString property can be used to save and load Xml contents of the formula at design time. At runtime it may also be used via a PropertyGrid control.

Access the XmlString property:



VariableNames	String[] Array
Visible	True
x_0	Form1.txtX0.Nun
x_1	Form1.txtX1.Nun
XmlString	<Root type="M
y_0	Form1.txtY0.Nu
y_1	Form1.txtY1.Nun

A dialogue box shows the Xml representation of the Math Expression. It does not allow editing manually because that will be too likely to make mistakes.



```
<Root type="MathNodeRoot" name="math" fixed="False"
editorRect="307,205,624,421"><Description></Description><ShortDesc></ShortDesc><IconIm
age><IconType type="EnumIconType">ItemImage</IconType><IconSize type="Size">32,
32</IconSize><TextAttributes><Font type="Font">Times New Roman, 12pt</Font><TextColor
type="Color">Blue</TextColor><BackColor
type="Color">White</BackColor></TextAttributes></IconImage><Position type="Point">0,
0</Position><Font type="Font">Times New Roman, 12pt</Font><SuperscriptFont
type="Font">Times New Roman, 8pt</SuperscriptFont><Inport
type="MathNodeVariableDummy" name="dummy" subscript="" asInport="True"
ID="1200752549"><ValueType type="RaisDataType"><LibType type="double"
/><Description></Description></ValueType><Port type="LinkLineNodeInPort"
portId="2844779262" instanceId="4028367822"><Location type="Point">0,
0</Location><PositionType
type="enumPositionType">Top</PositionType><Position>0</Position><LinkedPortID
linkedPortIndex="0">0</LinkedPortID><Label type="DrawingVariable"><Location
type="Point">0, 0</Location><Size type="Size">20, 20</Size><RelativePosition type="Point"
xTo0="True" yTo0="True">-20, -20</RelativePosition><LabelVisible
type="bool">False</LabelVisible></Label></Port></Inport><Math type="MathNodeVariable"
name="X" subscript="" asInport="False" ID="106202482"><ValueType
type="RaisDataType"><LibType type="double"
/><Description></Description></ValueType></Math><Math type="MathNodeSqrt"><Math
```

Copy -- Copy the xml contents to the Clipboard. Then the contents can be pasted to emails, text files, or other places as needed

Paste -- If others send Xml contents to you by email then the contents can be pasted here.

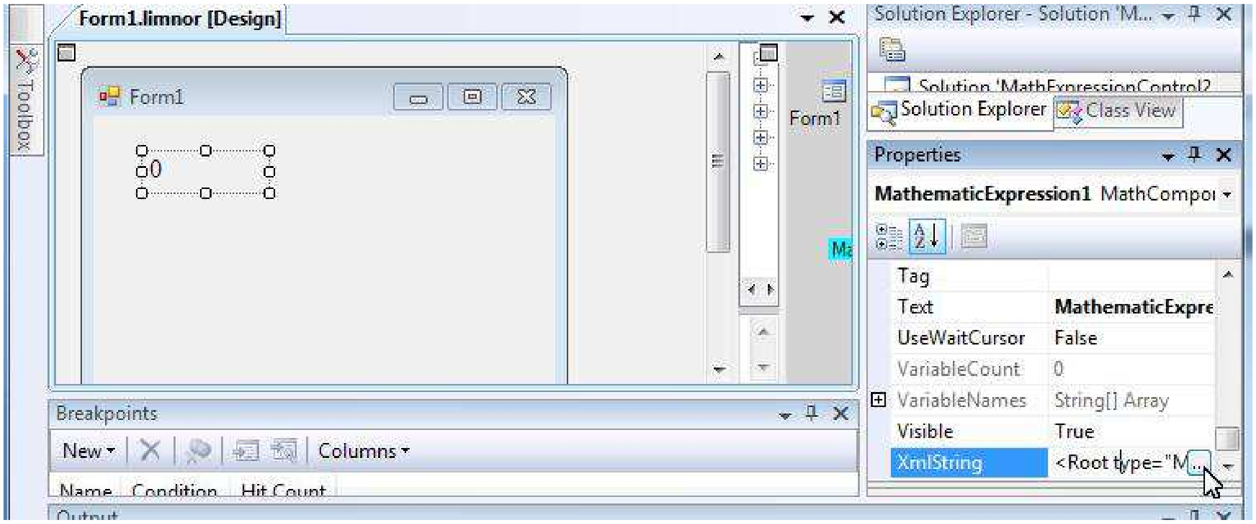
Save -- Save the current Xml contents to a file. It will display a save-file dialogue box for you specifying the file.

... -- Show an Open-File dialogue box to select a xml file to load. The file contents will be displayed in the text box

OK ✓ -- Close the dialogue box and use the xml contents in the text box for the Math expression.

Cancel ✗ -- Close the dialogue box and do not change the Math Expression.

Suppose we create another application and use a Math Expression Control which is empty. We access its XmlString property:

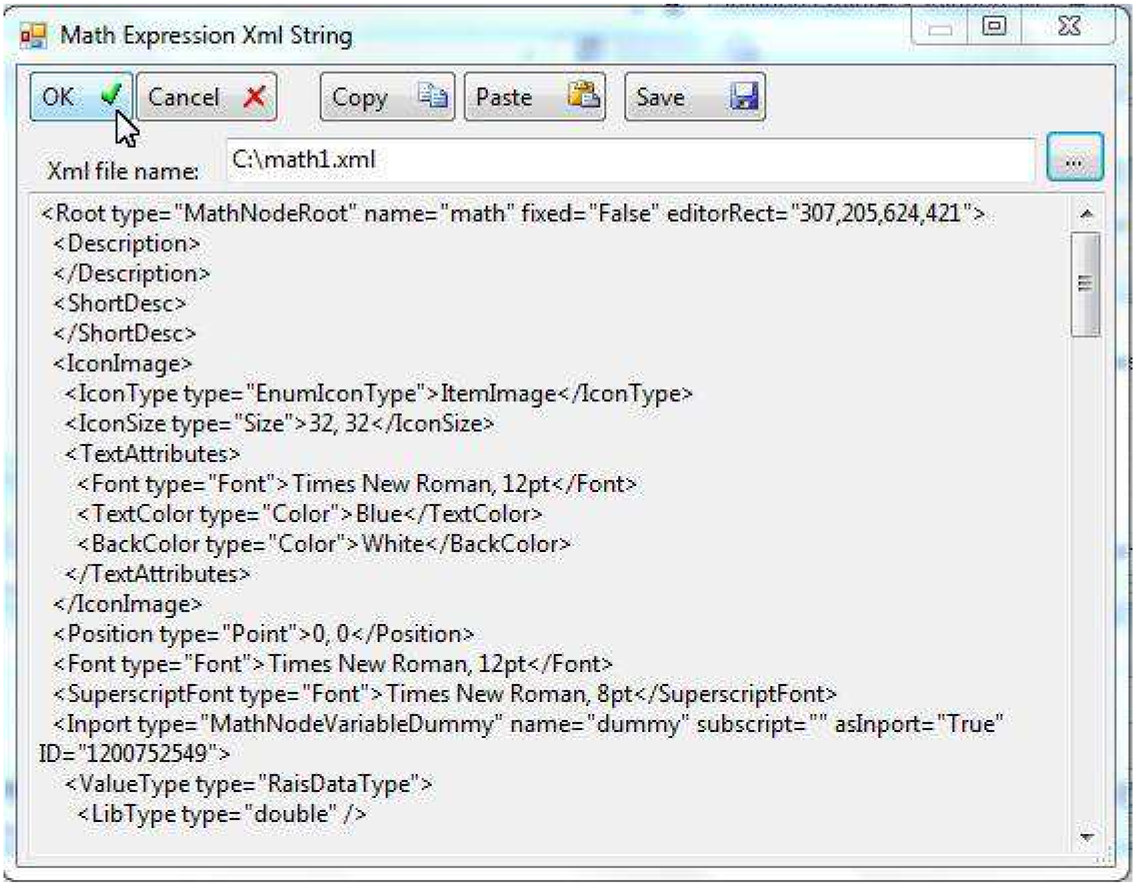


Properties

MathematicExpression1 MathCompo

Tag	
Text	MathematicExpre
UseWaitCursor	False
VariableCount	0
VariableNames	String[] Array
Visible	True
XmlString	<Root type="M...

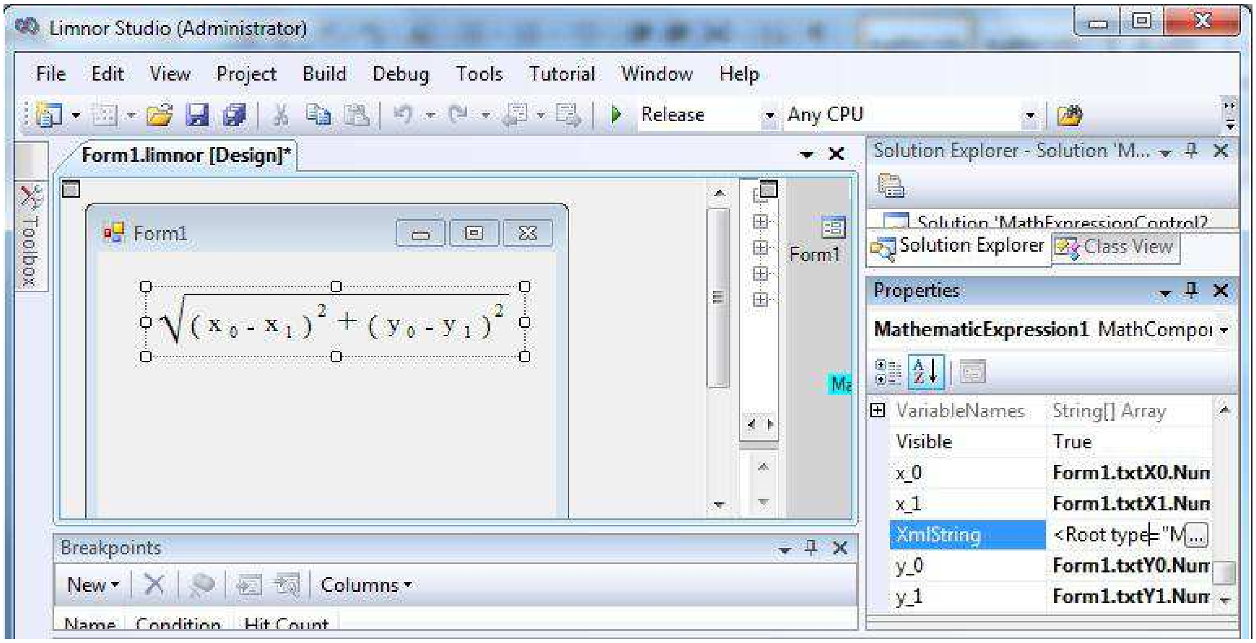
We created a xml file at c:\math1.xml before. Now we load this file by clicking ... . The xml contents are loaded into the text box:



Xml file name: C:\math1.xml

```
<Root type="MathNodeRoot" name="math" fixed="False" editorRect="307,205,624,421">
 <Description>
 </Description>
 <ShortDesc>
 </ShortDesc>
 <IconImage>
  <IconType type="EnumIconType">ItemImage</IconType>
  <IconSize type="Size">32, 32</IconSize>
  <TextAttributes>
   <Font type="Font">Times New Roman, 12pt</Font>
   <TextColor type="Color">Blue</TextColor>
   <BackColor type="Color">White</BackColor>
  </TextAttributes>
 </IconImage>
 <Position type="Point">0, 0</Position>
 <Font type="Font">Times New Roman, 12pt</Font>
 <SuperscriptFont type="Font">Times New Roman, 8pt</SuperscriptFont>
 <Inport type="MathNodeVariableDummy" name="dummy" subscript="" asInport="True"
ID="1200752549">
  <ValueType type="RaisDataType">
   <LibType type="double" />
```

Click OK. We have the Math Expression represented by the xml contents:

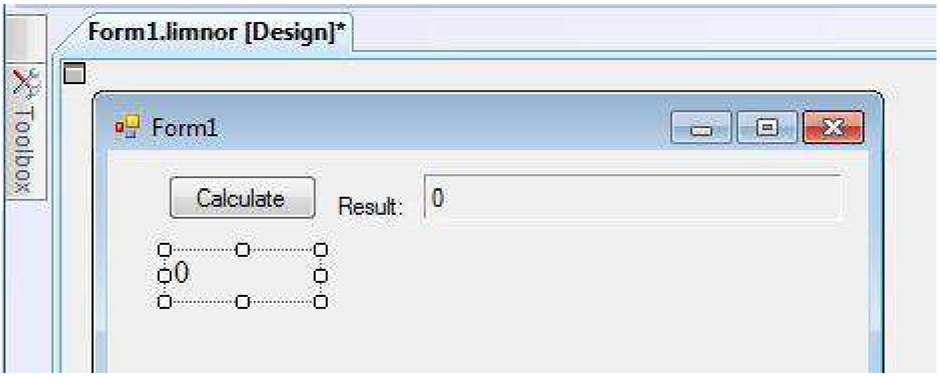

Limnor Studio (Administrator)
File Edit View Project Build Debug Tools Tutorial Window Help
Release
Any CPU
Form1.limnor [Design]*
Toolbox
Form1
$\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$
Solution Explorer - Solution 'M...
Solution 'MathExpressionControl2
Solution Explorer
Class View
Properties
Breakpoints
New
Columns
Name Condition Hit Count

MathematicExpression1 MathCompo

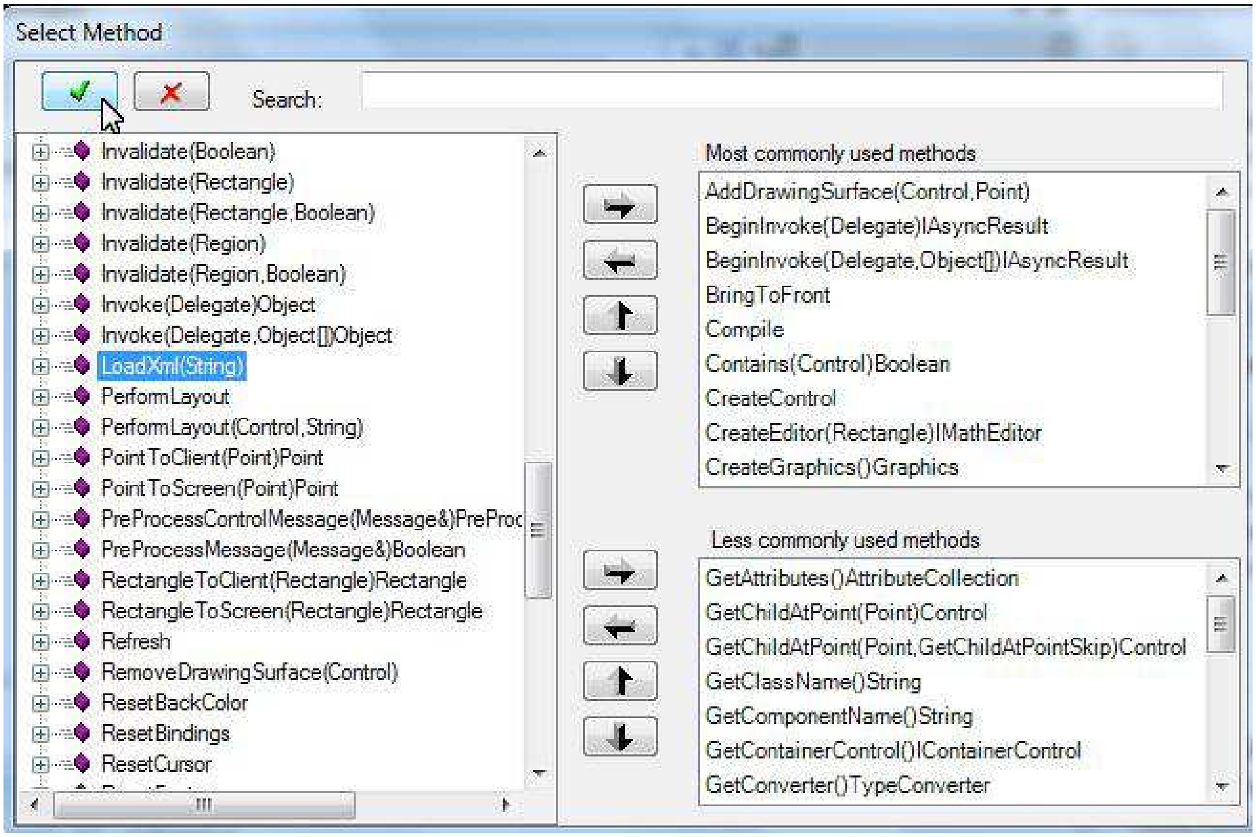
VariableNames	String[] Array
Visible	True
x_0	Form1.txtX0.Nun
x_1	Form1.txtX1.Nun
XmlString	<Root type="M
y_0	Form1.txtY0.Nun
y_1	Form1.txtY1.Nun

6.8 Load Xml at Runtime

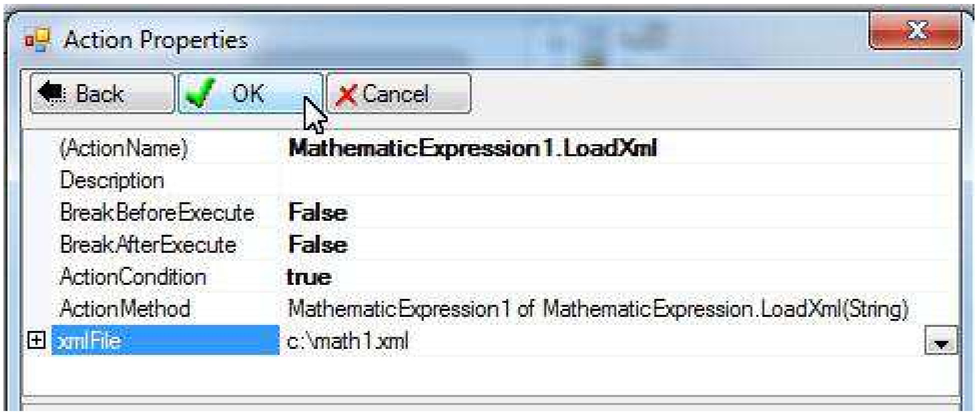
Create a new application and add a Math Expression Control to a form. As we did before we add a label for displaying calculation result. Create an Execute action and assign it to the Click event of a button.



LoadXml method allows us to load math expression from an xml file. Right-click the Math Expression Control; choose “Create Action”; choose “More Methods”; choose “All methods”. Select LoadXml:

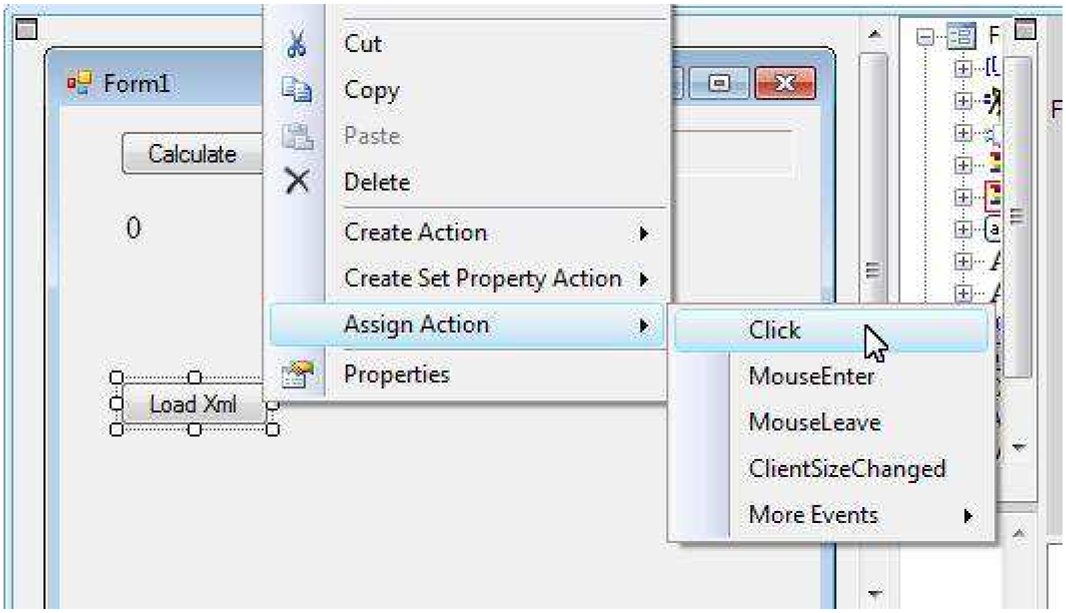

Select Method
Search:
Invalidate(Boolean)
Invalidate(Rectangle)
Invalidate(Rectangle,Boolean)
Invalidate(Region)
Invalidate(Region,Boolean)
Invoke(Delegate)Object
Invoke(Delegate,Object[])Object
LoadXml(String)
PerformLayout
PerformLayout(Control,String)
PointToClient(Point)Point
PointToScreen(Point)Point
PreProcessControlMessage(Message&)PreProc
PreProcessMessage(Message&)Boolean
RectangleToClient(Rectangle)Rectangle
RectangleToScreen(Rectangle)Rectangle
Refresh
RemoveDrawingSurface(Control)
ResetBackColor
ResetBindings
ResetCursor
Most commonly used methods
AddDrawingSurface(Control,Point)
BeginInvoke(Delegate)IAsyncResult
BeginInvoke(Delegate,Object[])IAsyncResult
BringToFront
Compile
Contains(Control)Boolean
CreateControl
CreateEditor(Rectangle)IMathEditor
CreateGraphics()Graphics
Less commonly used methods
GetAttributes()AttributeCollection
GetChildAtPoint(Point)Control
GetChildAtPoint(Point,GetChildAtPointSkip)Control
GetClassName()String
GetComponentName()String
GetContainerControl()IContainerControl
GetConverter()TypeConverter

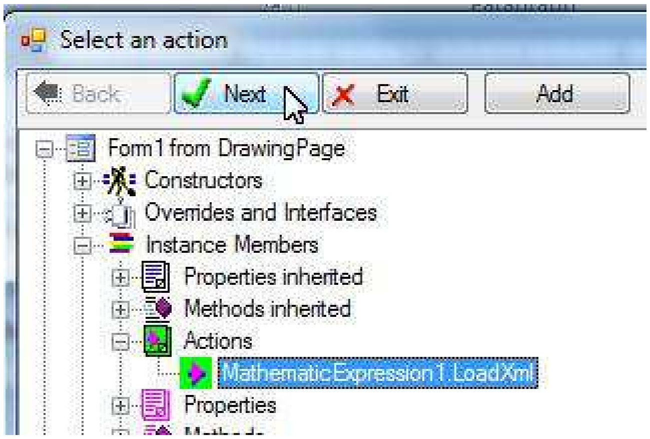
For simplicity, use a constant file name for this action:



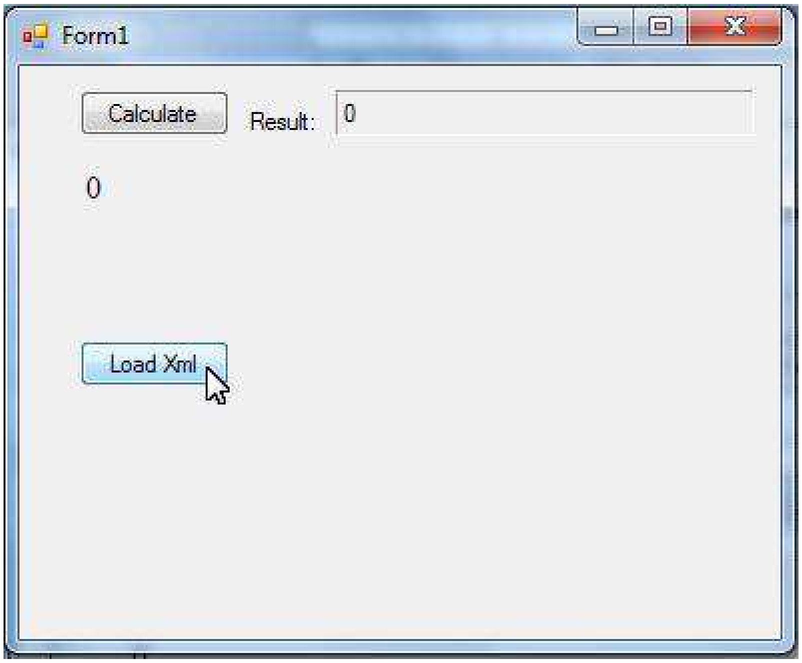
(ActionName)	MathematicExpression1.LoadXml
Description	
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	MathematicExpression1 of MathematicExpression.LoadXml(String)
xmlFile	c:\math1.xml

Assign the action to the Click event of a button:

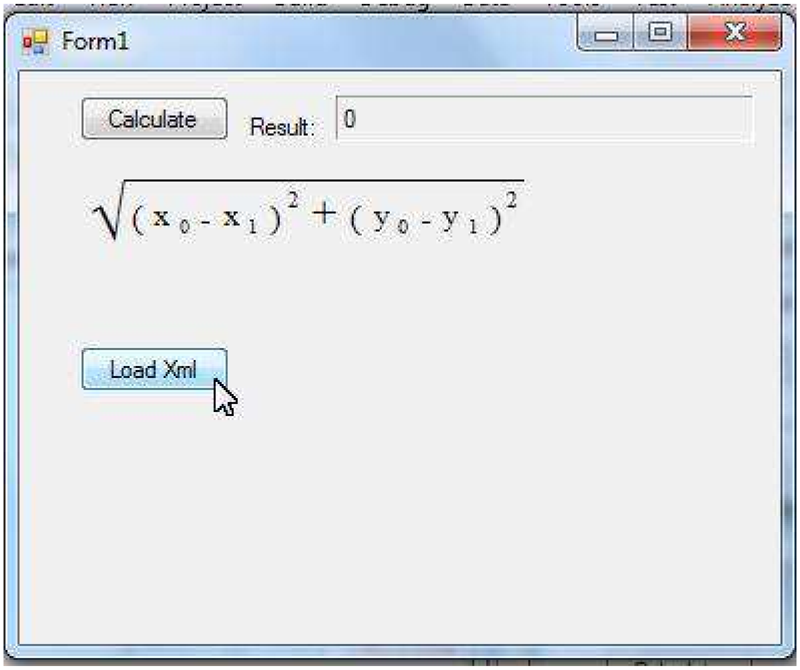




Run the application to test. Click the Load Xml button:

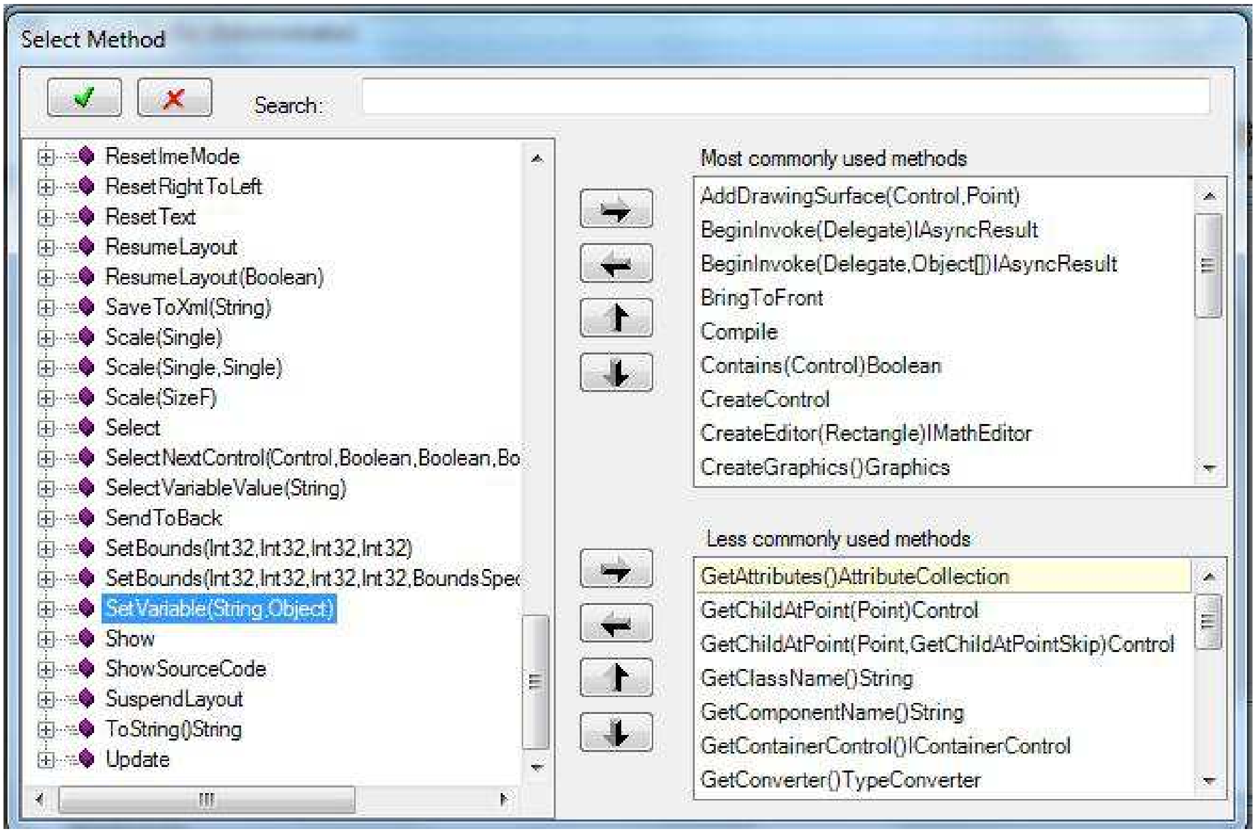


The formula represented by the Xml file displays:

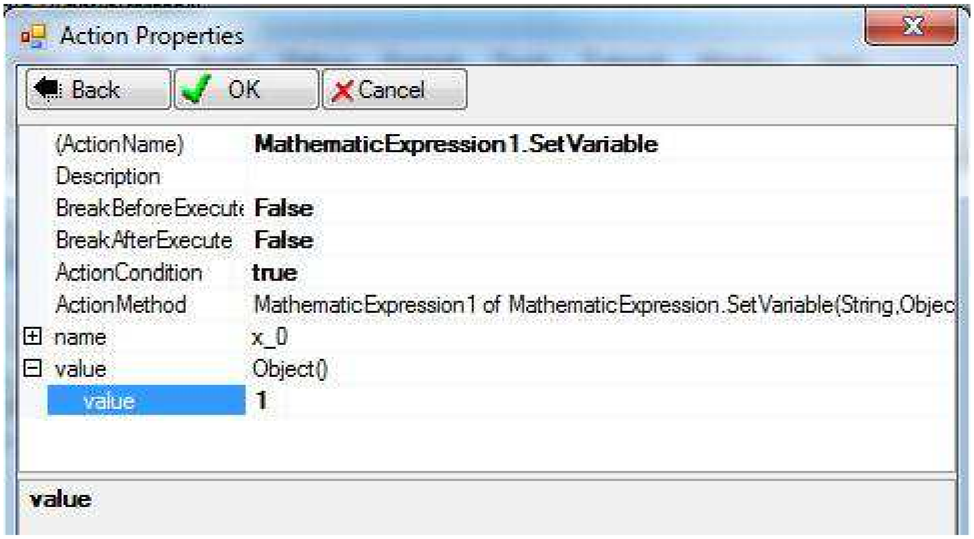


6.9 Set variable value by name

After we load the math expression from xml, we need give variables values before doing calculation. SetVariable method allows us to do that. Right-click the Math Expression Control; choose "Create Action"; choose "More Methods"; choose "All Methods". Select SetVariable method:


Select Method
Search:
ResetImeMode
ResetRightToLeft
ResetText
ResumeLayout
ResumeLayout(Boolean)
SaveToXml(String)
Scale(Single)
Scale(Single,Single)
Scale(SizeF)
Select
SelectNextControl(Control,Boolean,Boolean,Bo
SelectVariableValue(String)
SendToBack
SetBounds(Int32,Int32,Int32,Int32)
SetBounds(Int32,Int32,Int32,Int32,BoundsSpec
SetVariable(String,Object)
Show
ShowSourceCode
SuspendLayout
ToString()String
Update
Most commonly used methods
AddDrawingSurface(Control,Point)
BeginInvoke(Delegate)IAsyncResult
BeginInvoke(Delegate,Object[])IAsyncResult
BringToFront
Compile
Contains(Control)Boolean
CreateControl
CreateEditor(Rectangle)IMathEditor
CreateGraphics()Graphics
Less commonly used methods
GetAttributes()AttributeCollection
GetChildAtPoint(Point)Control
GetChildAtPoint(Point,GetChildAtPointSkip)Control
GetClassName()String
GetComponentName()String
GetContainerControl()IContainerControl
GetConverter()TypeConverter

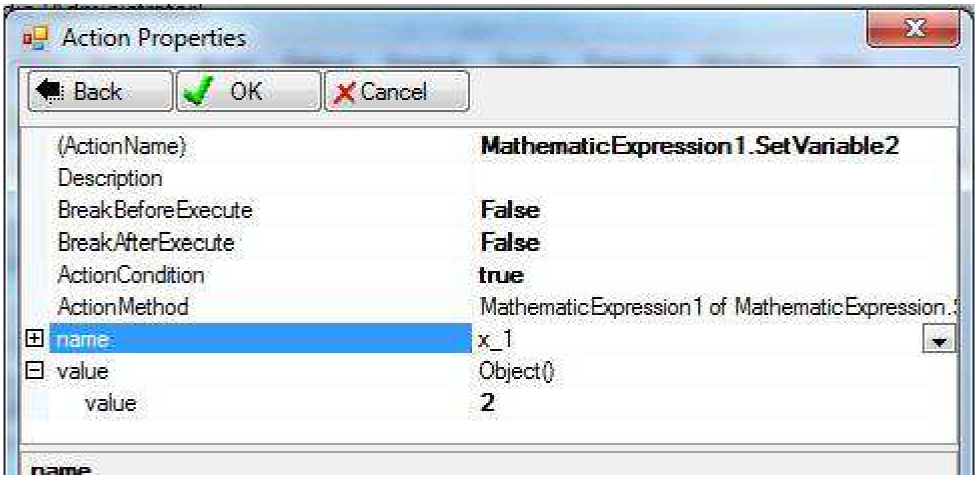
“name” property of this action is the name of the variable. x_0 is named as x_0, “value” property of this action is the value to be given to the variable:



(ActionName)	MathematicExpression1.SetVariable
Description	
BreakBeforeExecut	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	MathematicExpression1 of MathematicExpression.SetVariable(String,Objec
⊞ name	x_0
⊟ value	Object()
value	1

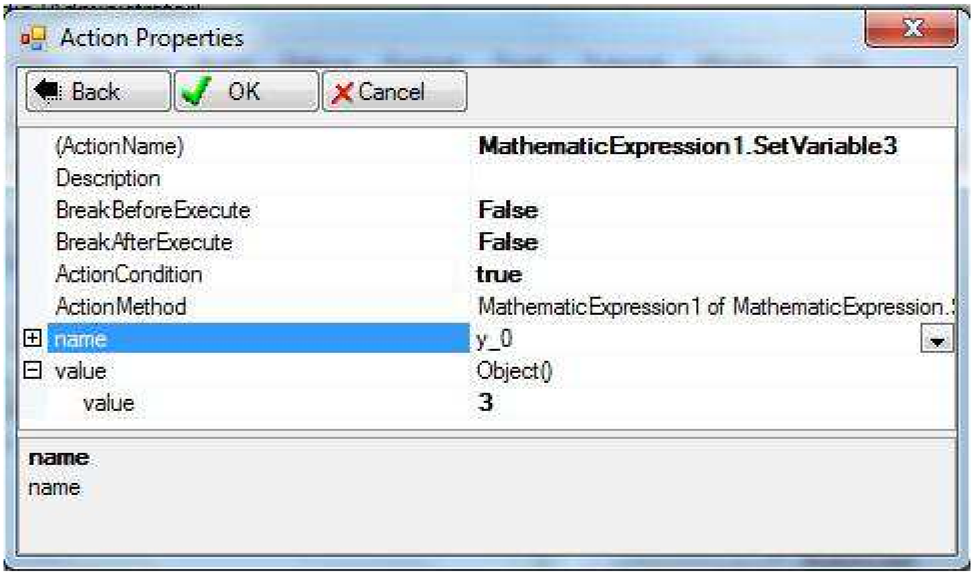
value

For each variable, create an action to give value:



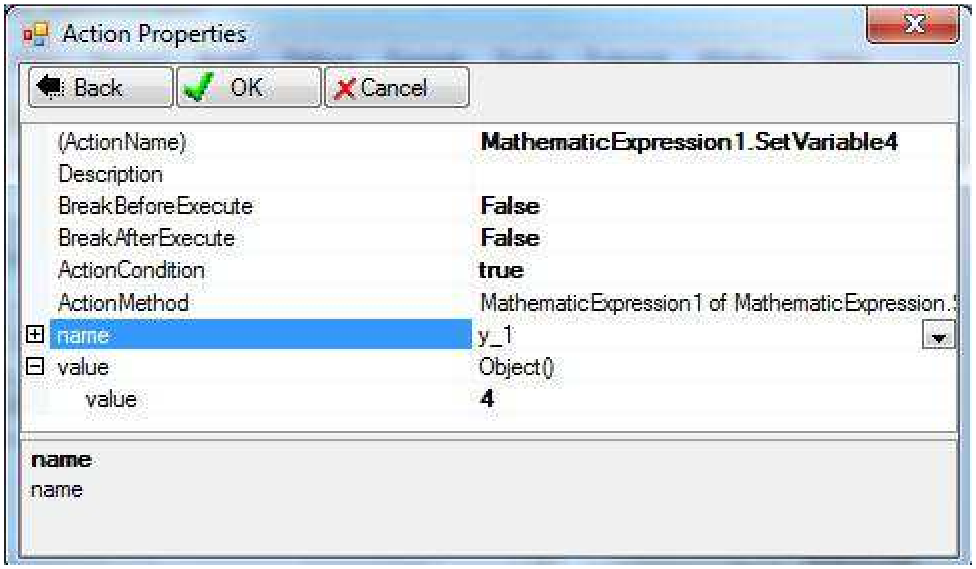
(ActionName)	**MathematicExpression1.SetVariable2**
Description	
BreakBeforeExecute	**False**
BreakAfterExecute	**False**
ActionCondition	**true**
ActionMethod	MathematicExpression1 of MathematicExpression.S
⊞ name	x_1
⊟ value	Object()
value	**2**

name



(ActionName)	**MathematicExpression1.SetVariable3**
Description	
BreakBeforeExecute	**False**
BreakAfterExecute	**False**
ActionCondition	**true**
ActionMethod	MathematicExpression1 of MathematicExpression.S
⊞ name	y_0
⊟ value	Object()
value	**3**

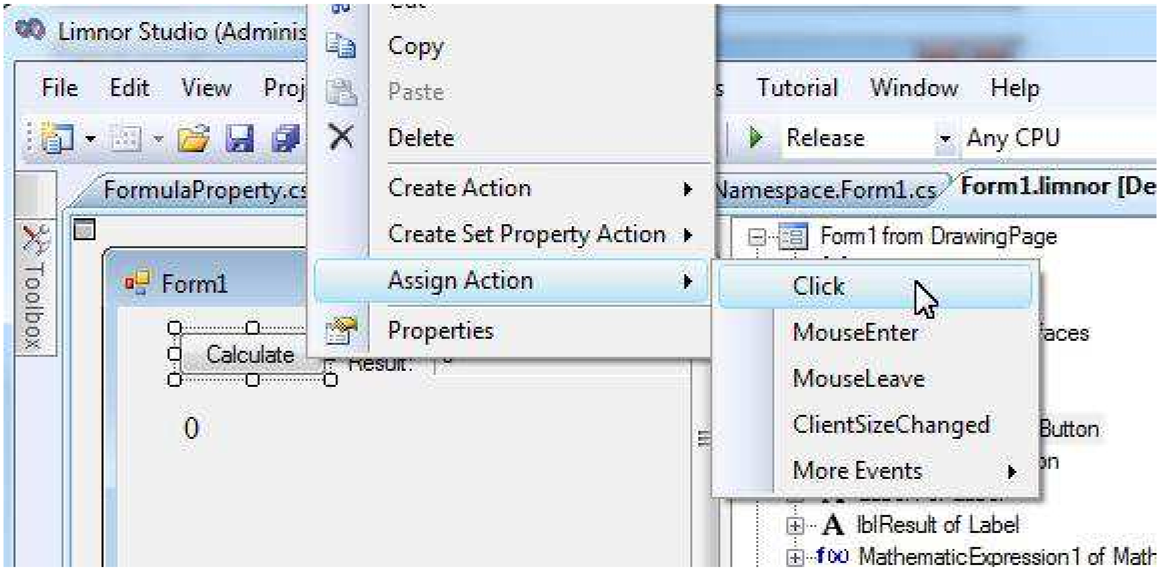
name
name



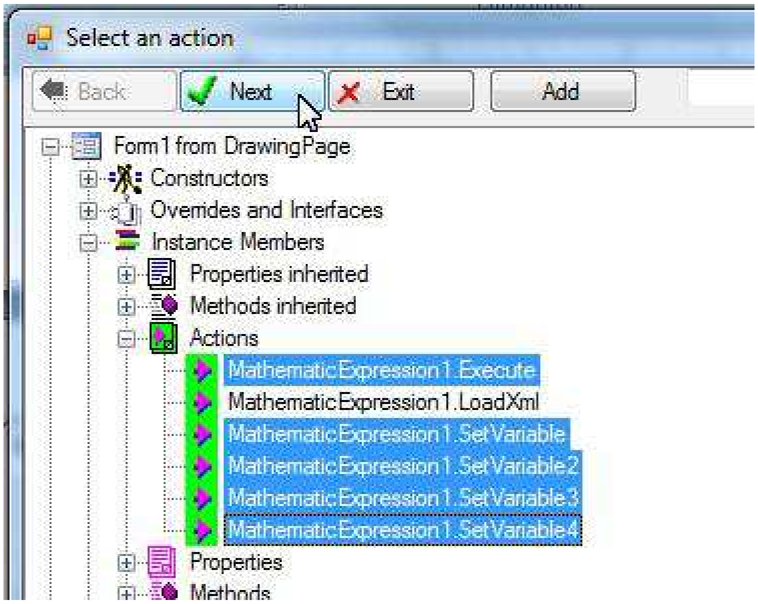
(ActionName)	**MathematicExpression1.SetVariable4**
Description	
BreakBeforeExecute	**False**
BreakAfterExecute	**False**
ActionCondition	**true**
ActionMethod	MathematicExpression1 of MathematicExpression.S
⊞ name	y_1
⊟ value	Object()
value	**4**

name
name

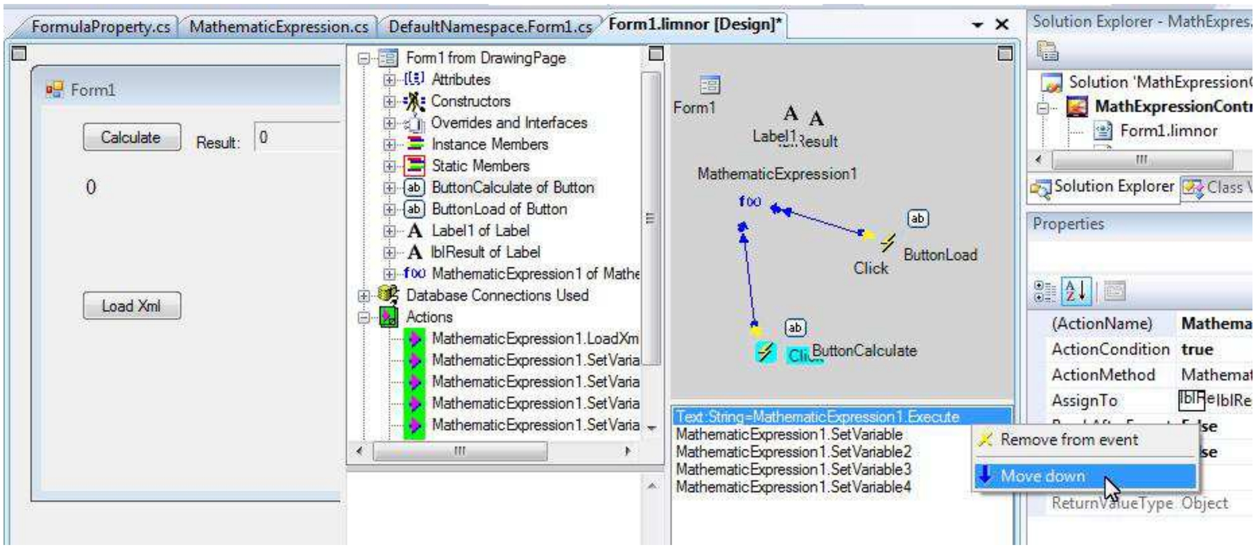
Assign the actions to the Click event of the Calculate button:

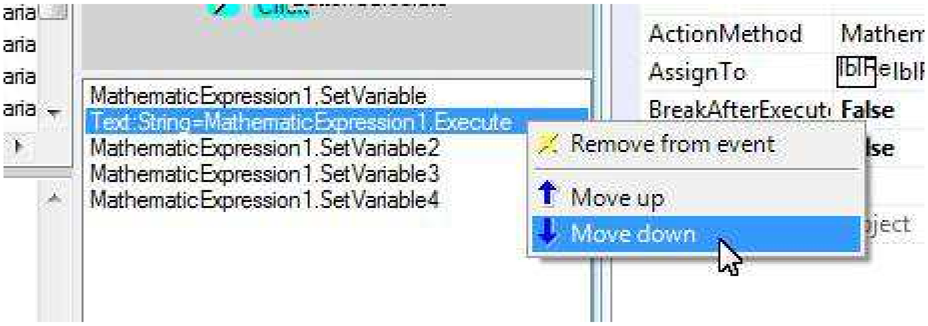


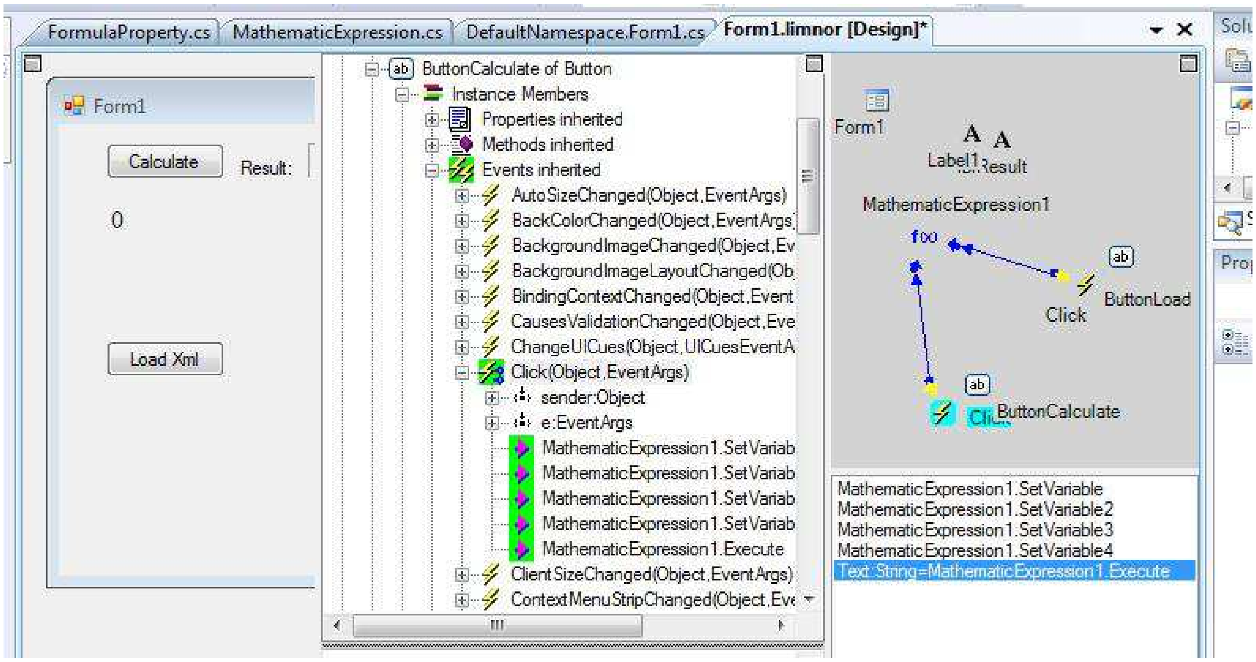
Select the actions to be used:


Select an action
Back
Next
Exit
Add
Form1 from DrawingPage
Constructors
Overrides and Interfaces
Instance Members
Properties inherited
Methods inherited
Actions
MathematicExpression1.Execute
MathematicExpression1.LoadXml
MathematicExpression1.SetVariable
MathematicExpression1.SetVariable2
MathematicExpression1.SetVariable3
MathematicExpression1.SetVariable4
Properties
Methods

We need to move the calculation action to the end of the action list for the Click event because the calculation should be done after all variables get their values:

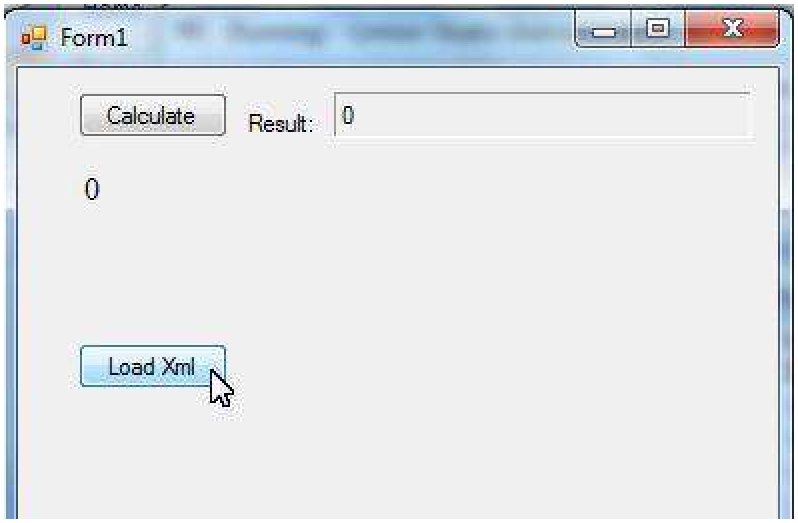



MathematicExpression1.SetVariable
Text:String=MathematicExpression1.Execute
MathematicExpression1.SetVariable2
MathematicExpression1.SetVariable3
MathematicExpression1.SetVariable4
Remove from event
Move up
Move down
ActionMethod Mathem
AssignTo lblI
BreakAfterExecut False


FormulaProperty.cs
MathematicExpression.cs
DefaultNamespace.Form1.cs
Form1.limnor [Design]*
Form1
Calculate
Result:
0
Load Xml
ButtonCalculate of Button
Instance Members
Properties inherited
Methods inherited
Events inherited
AutoSizeChanged(Object,EventArgs)
BackColorChanged(Object,EventArgs)
BackgroundImageChanged(Object,Ev
BackgroundImageLayoutChanged(Ob
BindingContextChanged(Object,Event
CausesValidationChanged(Object,Eve
ChangeUICues(Object,UICuesEventA
Click(Object,EventArgs)
sender:Object
e:EventArgs
MathematicExpression1.SetVariab
MathematicExpression1.SetVariab
MathematicExpression1.SetVariab
MathematicExpression1.SetVariab
MathematicExpression1.Execute
ClientSizeChanged(Object,EventArgs)
ContextMenuStripChanged(Object,Eve
Label1
Result
MathematicExpression1
Click
ButtonLoad
ButtonCalculate
MathematicExpression1.SetVariable
MathematicExpression1.SetVariable2
MathematicExpression1.SetVariable3
MathematicExpression1.SetVariable4
Text:String=MathematicExpression1.Execute

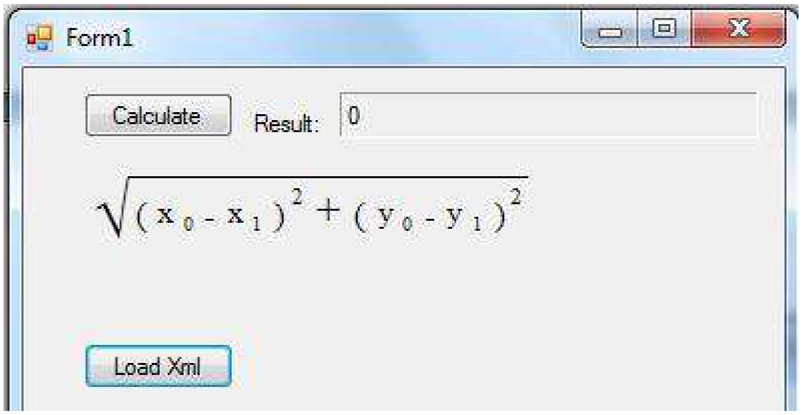
We may test it now.

Click Load Xml button to load the math expression from the xml file:



Result: 0

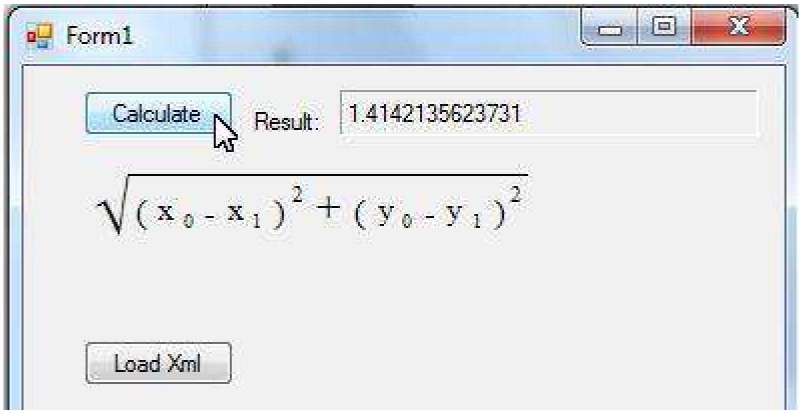
0



Result: 0

$$\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$$

Click Calculate button:



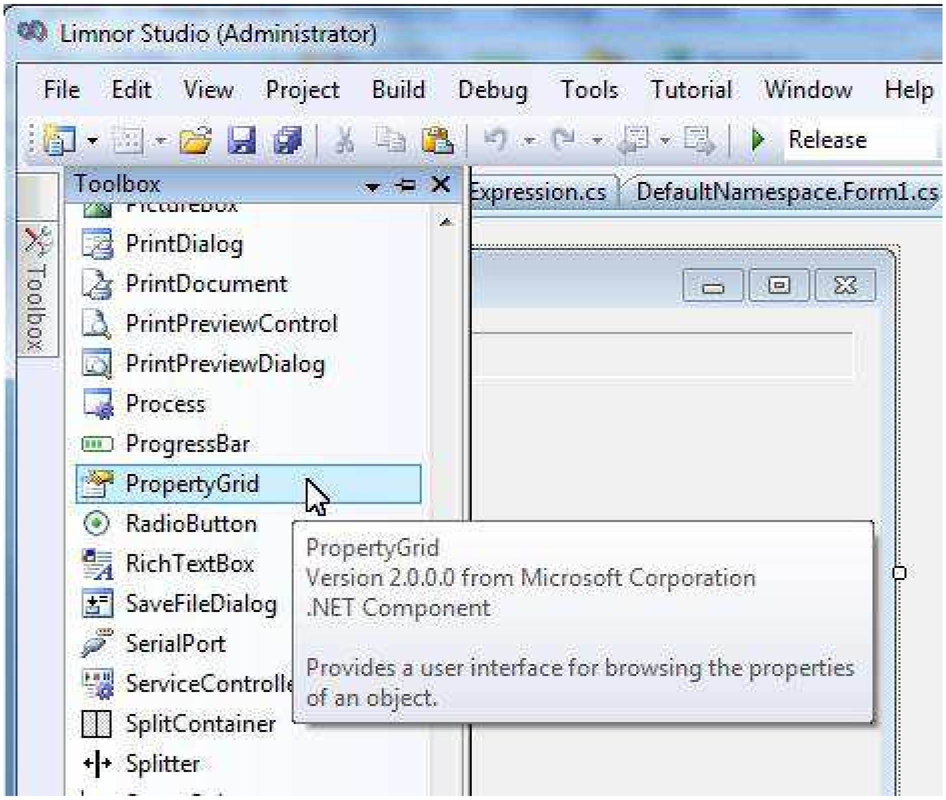
Result: 1.4142135623731

$$\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$$

6.10 Use PropertyGrid to give variables values

We need to know the variable names in order to use SetVariable actions. We may use a PropertyGrid to give variable values because a PropertyGrid tells us the variable names and allows us to give values.

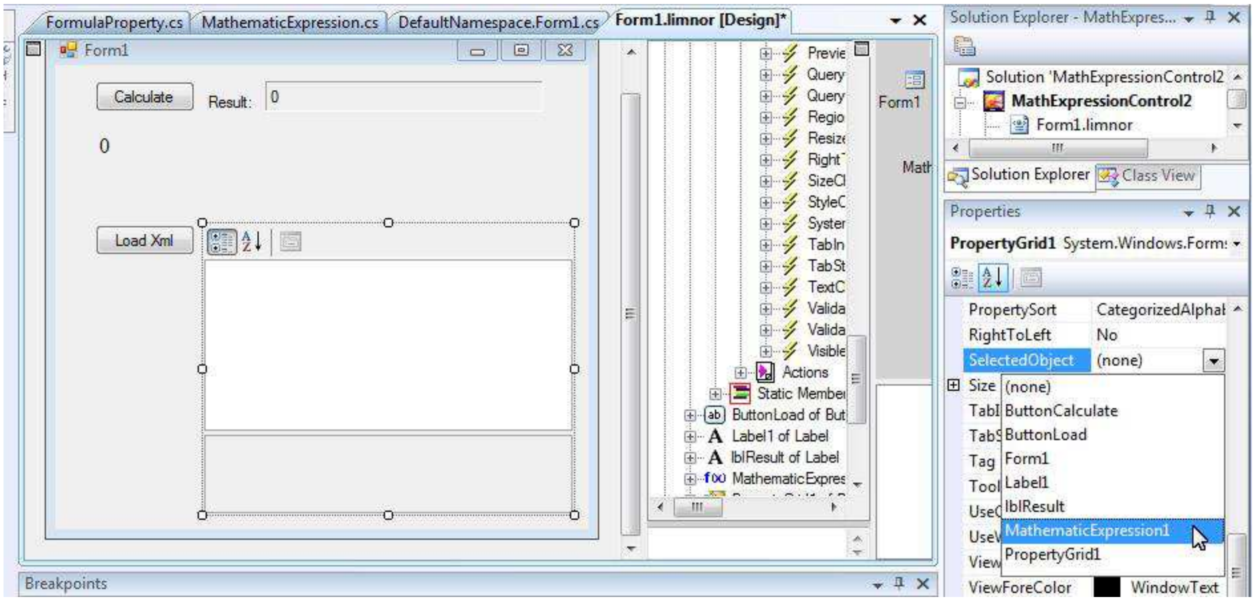
Add a PropertyGrid to the form:



PropertyGrid
Version 2.0.0.0 from Microsoft Corporation
.NET Component

Provides a user interface for browsing the properties of an object.

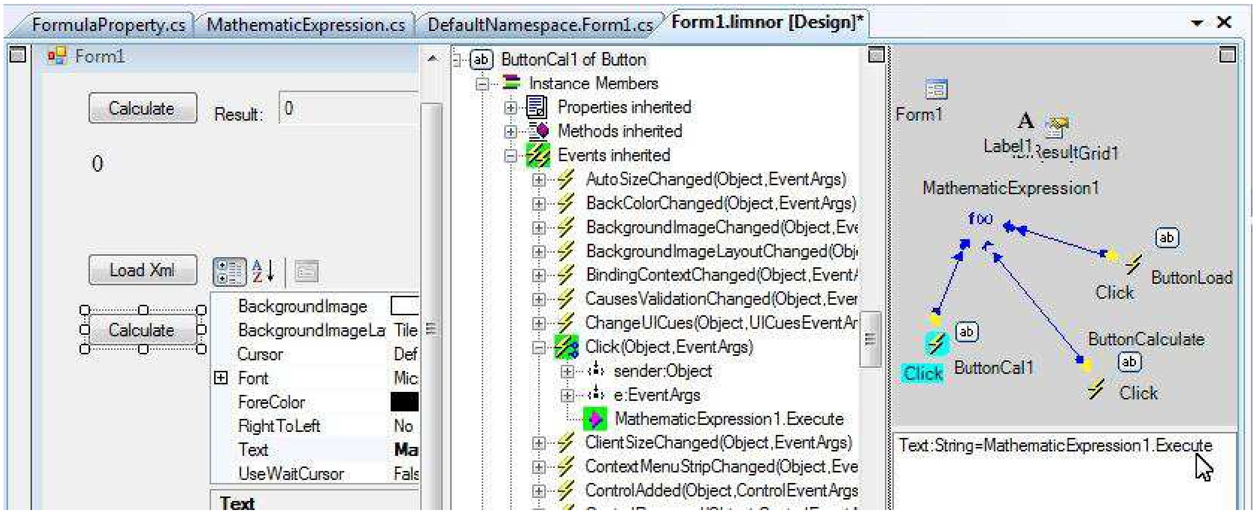
Set its SelectedObject to the Math Expression Control:



PropertyGrid1 System.Windows.Form:

PropertySort	CategorizedAlphal
RightToLeft	No
SelectedObject	(none)
Size	(none)
Tabl	ButtonCalculate
Tab$	ButtonLoad
Tag	Form1
Tool	Label1
UseC	lblResult
UseV	MathematicExpression1
View	PropertyGrid1
ViewForeColor	WindowText

Add another Calculate button. Only assign the Execute action to it:


FormulaProperty.cs
MathematicExpression.cs
DefaultNamespace.Form1.cs
Form1.limnor [Design]*
Form1
Calculate
Load Xml
Calculate
ButtonCal1 of Button
Instance Members
Properties inherited
Methods inherited
Events inherited
AutoSizeChanged(Object,EventArgs)
BackColorChanged(Object,EventArgs)
BackgroundImageChanged(Object,Eve
BackgroundImageLayoutChanged(Obj
BindingContextChanged(Object,Event/
CausesValidationChanged(Object,Ever
ChangeUICues(Object,UICuesEventAr
Click(Object,EventArgs)
sender:Object
e:EventArgs
MathematicExpression1.Execute
ClientSizeChanged(Object,EventArgs)
ContextMenuStripChanged(Object,Eve
ControlAdded(Object,ControlEventArgs
Form1
Label1
ResultGrid1
MathematicExpression1
f(x)
Click
ButtonLoad
ButtonCalculate
Click
ButtonCal1
Click

Result: 0

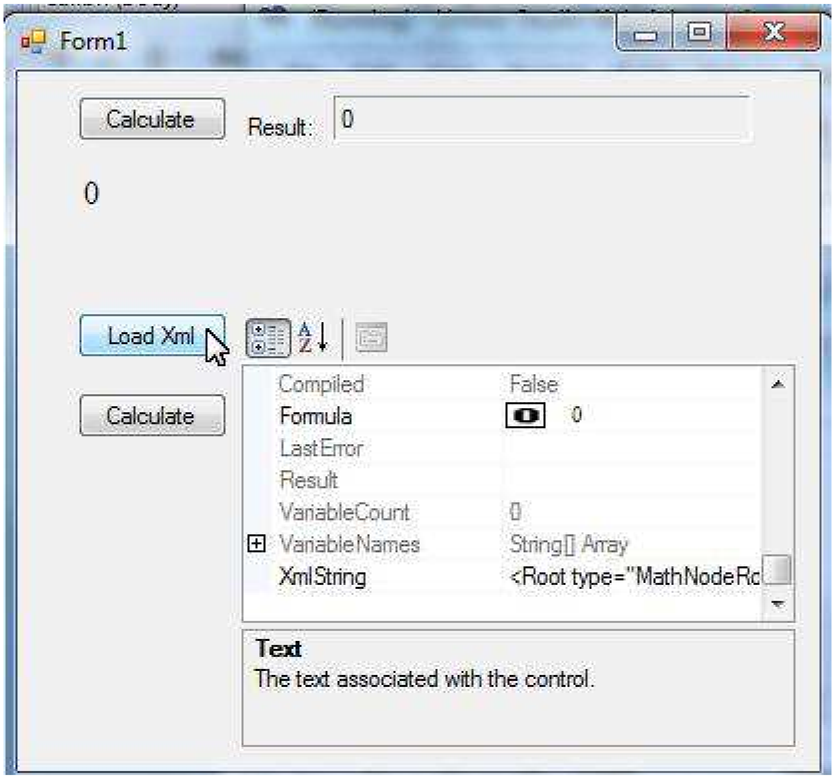
0

BackgroundImage	
BackgroundImageLa	Tile
Cursor	Def
Font	Mic
ForeColor	
RightToLeft	No
Text	Ma
UseWaitCursor	Fals

Text

Text:String=MathematicExpression1.Execute

Run the application. Click Load Xml button:



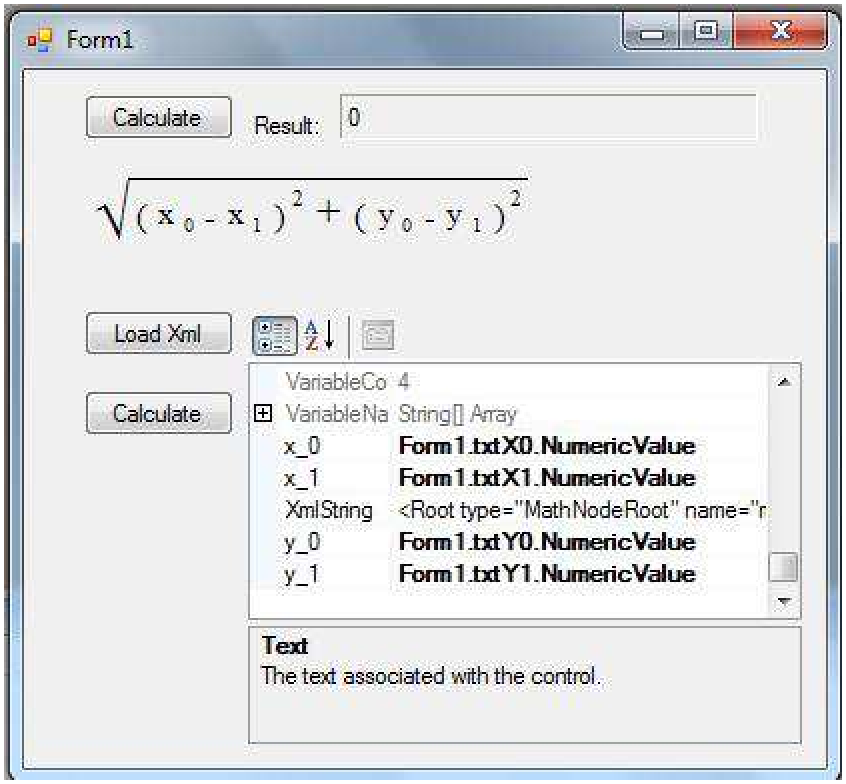
Result: 0

0

Compiled	False
Formula	0
LastError	
Result	
VariableCount	0
VariableNames	String[] Array
XmlString	<Root type="MathNodeRc

Text
The text associated with the control.

We can see that the variables appear in the PropertyGrid:

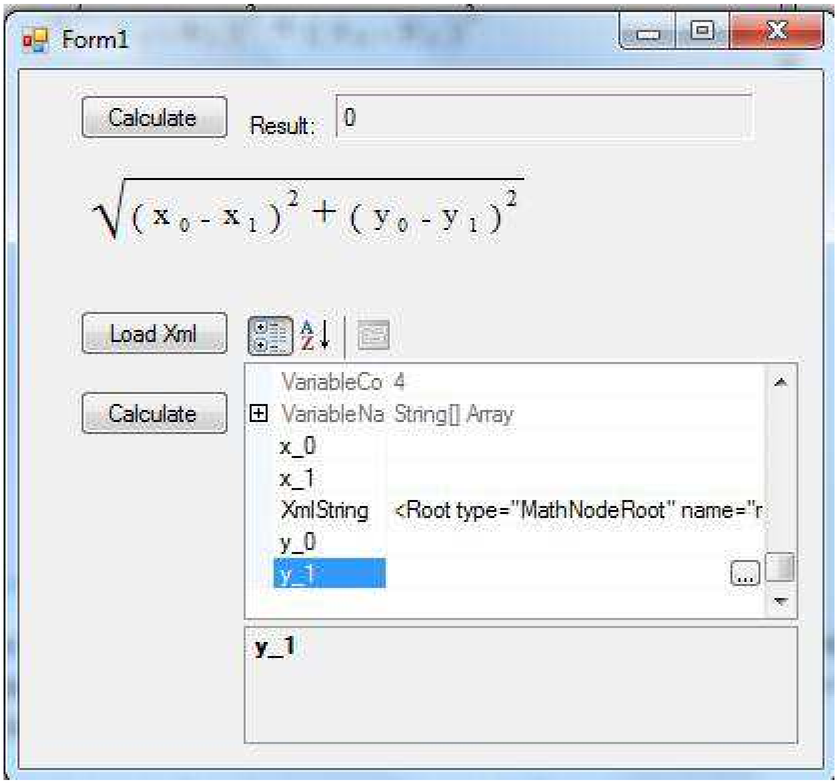


VariableCo	4
⊞ VariableNa	String[] Array
x_0	**Form1.txtX0.NumericValue**
x_1	**Form1.txtX1.NumericValue**
XmlString	<Root type="MathNodeRoot" name="r
y_0	**Form1.txtY0.NumericValue**
y_1	**Form1.txtY1.NumericValue**

Text
The text associated with the control.

Note that the variables show some value mappings because when the Xml file was generated the mappings were there. But those mappings are not valid in this application because we do not have those text boxes.

First, remove all those invalid mappings:

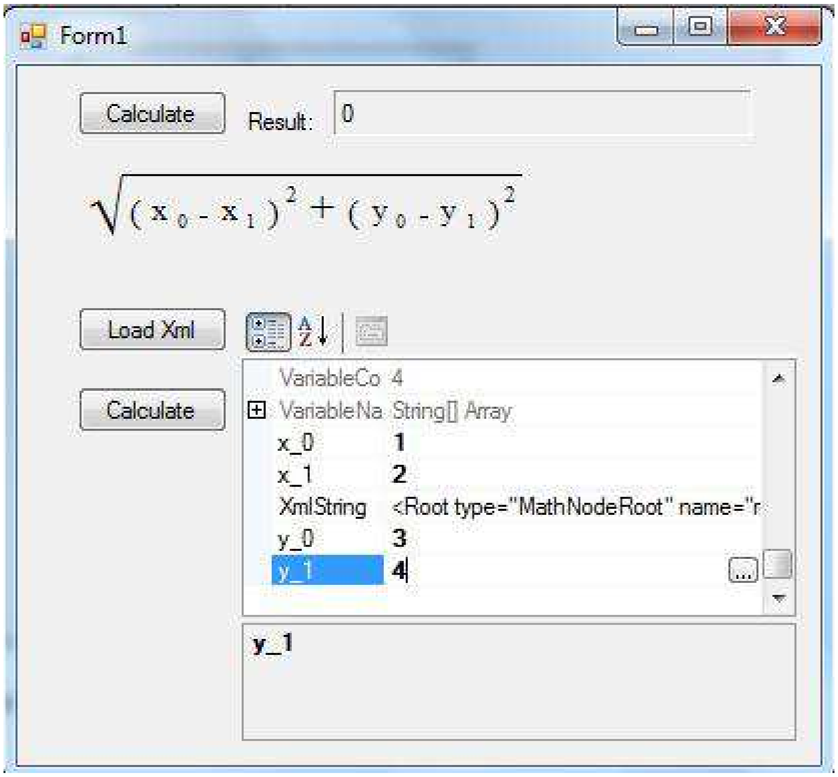


Result: 0

$$\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$$

VariableCo	4
⊞ VariableNa	String[] Array
x_0	
x_1	
XmlString	<Root type="MathNodeRoot" name="r
y_0	
y_1	

Type in the values we want to give to the variables:

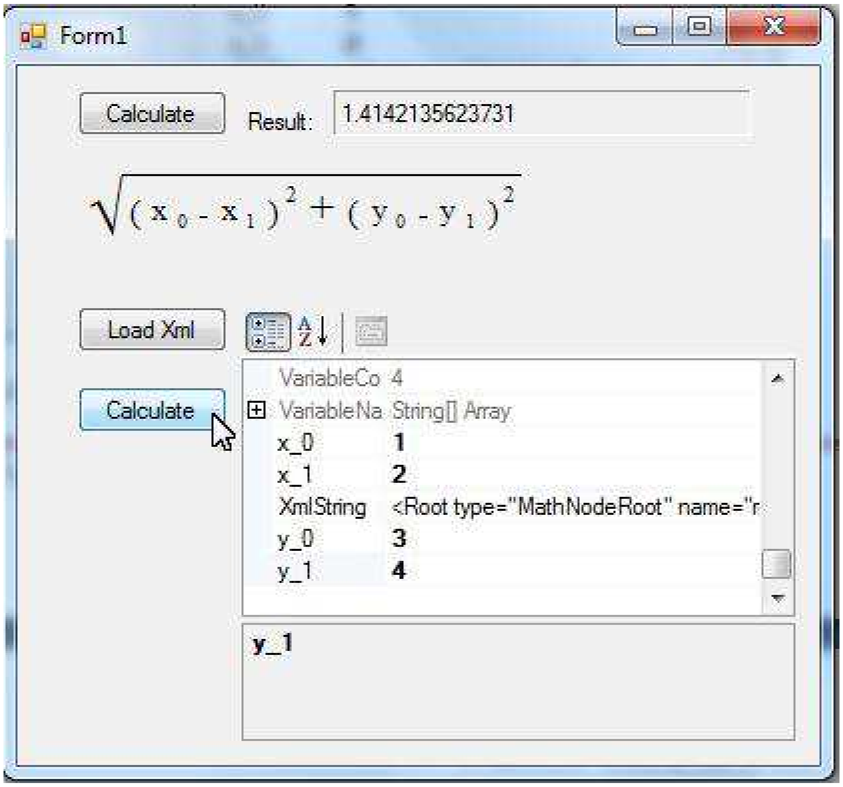


Result: 0

$$\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$$

VariableCo	4
⊞ VariableNa	String[] Array
x_0	1
x_1	2
XmlString	<Root type="MathNodeRoot" name="r
y_0	3
y_1	4

Click the new Calculate button; the calculation result appears:



Result: 1.4142135623731

$$\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$$

```
VariableCo  4
⊞ VariableNa  String[] Array
x_0         1
x_1         2
XmlString   <Root type="MathNodeRoot" name="r
y_0         3
y_1         4
```

y_1

7 Feedback

Please send your feedback to support@limnor.com.