

# LikeBasic

**Version 3.0**  
**Language Manual**  
© *WillmanSoft*  
*Finland 2011*

# 1. Introducing

Likebasic is a small script language for .NET framework (*tm Microsoft*) using BASIC syntax. Likebasic engine component has features for integrating language as part of the other programs. There are very few library commands inside the language but it is using .NET classes and libraries with endless options. Any application specific commands for solution could be added by a host program.

There is a small demonstration program 'LikeBasic Console', which allows testing of the engine and running scripts. Source code for LikeBasic Console could found in Form1.\* and likelib.cs source files.

There is also included 'Load' command in LikeBasic Script engine to include any class libraries from .NET framework or any other library directly.

LikeBasic is case insensitive including calls to the .NET framework. It has flexible type conversion system like many script languages.

Any variable could be integer, boolean, real, string, array or object only depending of previous assignment. Type is tried to be converted to correct type at runtime when making calls to .Net methods. New for version 2.0 are fixed array for some framework calls, which could be created using [ ] array operator.

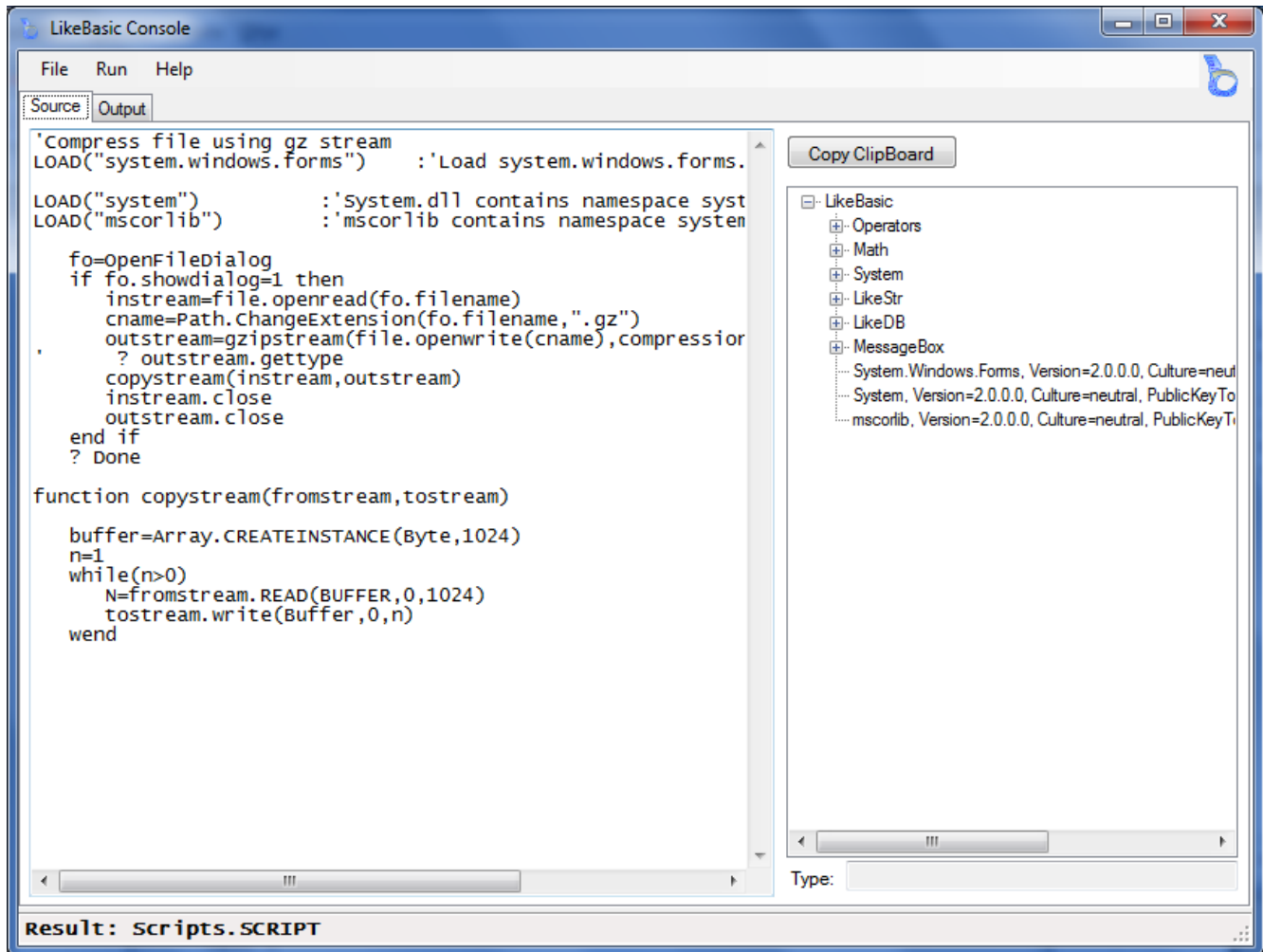
Version 2.0 presents new parameter ranking for overloaded functions, to find best function call based on parameter values.

Dynamic arrays are simply by storing values to index of new variable.

Language is structural, so is does not support old fashion goto or gosub statements. Code is written by using branching, loops and functions.

Language is using .NET garbage collection. Unused variables are released with garbage collector.

LikeBasic is best in small scripts, where the simplicity of source code is a primary need.



## 2. Statements

There is two kind of statements: store value or function. There is also possible retrieve value of variable, without doing anything with that. Because functions and variables are not separated at interpretation phase, error could not been detected. Statement ends at the end of the line or to ':' character, which allows to type multiple commands in one line.

Because none of the structures, including IF structure, won't end at the end of the line, there is no need to put multiple commands to single line. There is also no change in execution time. Only reasonable usage might be adding the comment at the end of the line.

Example of store statements

```
A=1:B="a2"  
C(A)=B
```

Example of function calls. Assume that we have implemented "write" call :

```
WRITE(dev,str)  
WRITE
```

## 3. Commenting

End of the single line could be commented with 'rem' word or using ' character. If there is command before rem there must be ':' character before 'rem'.

Sample:

```
Rem this is a comment!  
' Comment continues  
A=1 :set A to value 1
```

## 4. Types

Because types are assigned to variable at run time, so there is no syntax for declaring the type of variable. Internally there are following types:

**Boolean** is result of compare operation. Numeric value other than zero is true. Also string value True is handled as true boolean. Value is stored as integer value 1.

**Integer** is 64 bit length and it is used for bitwise operations.

*Examples of integer constants: 1, 123*

**Real** is used for arithmetic operations.

*Examples of real constants: 1.0, 123.5*

**Datetime** is an object. It could be used for calculation like type of Real.

Example: `datetime.parse("10/23/2009")+5`

**String** is a dynamic string. String constants are delimited with " or ' character at begin and end. Same character must be used.

*Example of string constants: "abcd", "123", 'Other delimiter', 'text' '*

**Array** is a dynamic list of values indexed with key value. Key of list could be any basic type except objects. It could be object only when the value of object could be converted to some basic type.

**Object** could be any .NET class type.

If variable is newer initialize then the value of variable is the name of the variable just how it was written in first place. This is useful because variables could be used as enum values without string.

**Static Array** is handled as object.

## 5. Operators

Listed in order of precedence. Begin with lowest order of precedence.

**= (Let)** is used store the value into the variable. Operation must be after variable or variable dimension.

**,** (**Comma**) Append item to the end of the list. Read further about arrays.

**OR** Logical or operation using short circuit. Arguments are two boolean values

**AND** Logical and operation using short circuit. Arguments are two boolean values

**=, <, >, <>, <=, =>** comparison operation, which are returning Boolean value. Parameter could be numeric or strings. If both are numeric then compare is made using numeric value. If one or both value is string then both parameters are converted to string before compare.

**;** (**Semicolon**) is used for concatenating strings.

**| (Pipe)** , **~ (tilde)** is used for binary or and xor. Arguments are two integers

**& (and)** for binary and. Arguments are two integer values

**+(plus)**, **-(minus)** are for addition and subtraction. Arguments are two real values

**\*(asterisk)** / are for multiply and division Arguments are two real values

**^ (cap)** for exponent. Arguments are two real values

**.** (**dot**) field operation. Left side is object and right side is field or method name.

Execution order could be changed with parenthesis.

## 6. Unary operations

**-(Minus)** Negation of numeric value.  
**Not** Inverse Boolean operation

## 7. Control Structures

### Branching (IF)

'If' -statement is typical basic structure with minor changes. Notice that there is always 'end' statement at end of the structure. 'If' word after 'end' is optional.

#### Syntax

```
IF <Condition> THEN
    <Statements if condition true>
[ELSE
    <Statements if condition false>]
END [IF]
```

Example:

```
IF x>2 THEN
    x=2
ELSE
    x=0
END IF
```

### For Loop

'For' loop is quite standard in all basic implementations. Statements inside of structure might be executed 0 to n times depending on start and end values. Loop variable is incremented or optional decremented with step value until end value is reached.

Loop variable is always defined as local variable, like it was defined using dim.

If 'step' statement is omitted then number 1 is used for increment.

Also the end value will be executed in loop.

#### Syntax

```
FOR <loop variable>=<startvalue> TO <endvalue> [STEP <step>]
    <statements>
NEXT [<loop variable>]
```

Example

```
FOR i=1 To 10
  a=a*i
NEXT I
```

## **While Loop**

While loop is executed until the condition is false. If value is false at start then loop won't be executed.

*Syntax:*

```
WHILE <Condition>
  <Statements while condition true>
WEND
```

Example

```
a=1
While a<100
  a=a*2
Wend
```

Function definitions

Function definition starts with 'function' keyword. Function ends at the end of the file or to another function keyword. Function definition is always global.

*Syntax:*

```
Function <name>[(parameter1{,parameter2})]
  <Statements>
{Function <name>[(parameter1{,parameter2})]
  <Statements>}
```

*Example:*

```
function myfun

  return 0

function myfun2(a,b)

  return a*b+1
```

## 8. Arrays

### *Dynamic arrays*

Dynamic array is list of values which are referred by key value. Key value could be any basic type including array itself. Special arrays called lists are indexed by raising sequence of numbers starting from number one. They could be used as parameter lists for function calls, when there is need to give more than one argument.

#### *Syntax*

```
<List variable>(<Key value>)
```

```
<List variable>(<Key value>)=<value>
```

```
<Value1>,<Value2>{,<Valuen>}
```

#### *Example*

```
A(1)=1
A("test")=3
B=2,3
?A("test")
?B(2)
```

Output:

```
3
3
```

### *Static arrays [ ]*

Static arrays are for calls to external system classes. Static arrays are typed. They are created by using array operator [ ]. This is only for construction, accessing will be made using standard parenthesis ().

Syntax:

```
<variable>=<type>['<index1>[,<index2>[,<indexn>]]']
```

Sample

```
myarray= string[5]
mybarray = byte[80,25]
```

## 9. Function calls

Function calls are user defined functions or calls to static methods in .NET classes. Function call without return values are considered as procedure calls.

*Syntax*

*<Function name>[(*<parameters>*)]*

Example

a=MyFun(3)

MyFun2(4)

MyFun3

## 10. Object fields and methods

Object could be created by calling function by its name of class. Object fields and methods could be accessed with field operand. Arguments are converted dynamically to the destination type. Object fields are used as like all other variables.

Syntax

<Object variable>.<Method>[(parameters)]

| <Object variable>.<Field>[(parameters)]

Example

PARETFORM.Caption=PARETFORM.Caption;"- My Caption"

## 11. Local variable (DIM)

There is possible to define local variable in functions by using DIM word. This is useful feature if program is little bit bigger, or there is need to make recursive functions. Loop variable in 'for' loop is always define as local variable.

Syntax

```
DIM <Identifier>{,<Identifier>}
```

Example

```
DIM a,b,c
```

## 12.Function result

Function result could be returned by using return procedure. It could return any type of data. Return without value just ends the function and continues execution in calling program.

Return could also be used in main for ending it. Return could be shortened as equal sign ('='). This may be useful when creating simple calculated values.

*Syntax:*

*Return*

```
| Return <Value>  
| = <Value>
```

Example

```
= 1
```

```
Return 1
```

```
--
```

## 13. Print procedure

Print function is used for testing results of software at development time. It is shortened by '?' character. Host application must implement the printing the data to device, file or to screen.

Print command output data as string. If there are multiple parameters they are printed by separating values with TAB control code ascii code 9.

If the statement end to semicolon ';' then data is printed with out line feed ascii 13 and 10. Otherwise they are added to data.

### *Syntax*

```
PRINT <Parameters>[:]  
? < Parameters>[:]
```

### Example

Print "Value of variable a is";a

```
? "-----";
```

## 14. Creating new object

LOAD method will load the library and import constructors as functions. Calling the name of class will call constructor and creates new instance of class. Notice that dll name is actually the name of a file, not a namespace, even when sometimes they are same. If no path has not been not specified, default framework library path will be used or secondarily dll will be search from application directory.

'New' function in version 1.0 has been removed.

### Syntax

```
Load <dll name[.dll]>
<var>=<constructor function>[(<parameters>)]
```

### Example

```
Load("system.windows.forms")
F=FORM
f.show
```

```
Load(".\locallib.dll")
F=localclass(1)
```

Some common dll files

mscorlib.dll - System, System.IO

system.dll - System.Compress

system.drawing.dll – System.Drawing

system.windows.forms.dll – System.Windows.Forms

Standard names of dlls could be found from .Net framework SDK or Visual Studio documentation.

## 15. Handling Event

Internal function 'EVENT' can be used to create event handler from LikeBasic function. It has one parameter: function of LikeBasic.

*Syntax:*

```
<eventhandler>=event(<Function>)
```

*Sample of using Events with Views:*

```
select=event(OnSelect)
```

```
function OnSelect(sender,ex)
  DataSet=SQL("SELECT * FROM table")
  Return
```

*Sample of using Events with Mouse:*

```
shape1.MouseDown=event(OnMouseDown)
```

```
function OnMouseDown(sender,ex)
  ShowMsg("CLICK")
  Return
```

## 16. Libraries of LikeBasic

This is a list of LikeBasic libraries. There is basic string library to have standard Basic string actions. Math library and MessageBox are imported directly from .NET library. LikeDB, LikeIO and DBEXlib are imported from DBEXform. LikeConsole is primarily for demonstration usage to show how to extend language in other applications.

### **LikeStr**

Likestr source is located in Likelib.cs file. There is also possibility to use standard .NET strings. In this library functions are for basic style strings . Major difference is that in c# string starts from 0 but in basic the first position is 1.

```
public static int Len(string s);
public static int Asc(string s);
public static int InStr(string s,string s);
public static int InStr(string s,string s, int from);
public static string Chr(int i);
public static string Format(format, arg1,arg2,... argn);
public static string Mid(string s,int start,int len);
public static string Mid(string s,int start);
public static string Left(string s,int len);
public static string Right(string s,int len);
public static string Trim(string s,int len);
public static string LTrim(string s,int len);
public static string RTrim(string s,int len);
public static string Upper(string s);
public static string Lower(string s);
```

### **Math**

Math library is directly imported from system namespace. Documentation for this could be found in .NET documentation.

```
public static decimal Abs(decimal value);
public static double Abs(double value);
public static float Abs(float value);
public static int Abs(int value);
public static long Abs(long value);
public static sbyte Abs(sbyte value);
public static short Abs(short value);
public static double Acos(double d);
public static double Asin(double d);
public static double Atan(double d);
public static double Atan2(double y, double x);
```

```

public static long BigMul(int a, int b);
public static decimal Ceiling(decimal d);
public static double Ceiling(double a);
public static double Cos(double d);
public static double Cosh(double value);
public static int DivRem(int a, int b, out int result);
public static long DivRem(long a, long b, out long result);
public static double Exp(double d);
public static decimal Floor(decimal d);
public static double Floor(double d);
public static double IEEERemainder(double x, double y);
public static double Log(double d);
public static double Log(double a, double newBase);
public static double Log10(double d);
public static byte Max(byte val1, byte val2);
public static decimal Max(decimal val1, decimal val2);
public static double Max(double val1, double val2);
public static float Max(float val1, float val2);
public static int Max(int val1, int val2);
public static long Max(long val1, long val2);
public static sbyte Max(sbyte val1, sbyte val2);
public static short Max(short val1, short val2);
public static uint Max(uint val1, uint val2);
public static ulong Max(ulong val1, ulong val2);
public static ushort Max(ushort val1, ushort val2);
public static byte Min(byte val1, byte val2);
public static decimal Min(decimal val1, decimal val2);
public static double Min(double val1, double val2);
public static float Min(float val1, float val2);
public static int Min(int val1, int val2);
public static long Min(long val1, long val2);
public static sbyte Min(sbyte val1, sbyte val2);
public static short Min(short val1, short val2);
public static uint Min(uint val1, uint val2);
public static ulong Min(ulong val1, ulong val2);
public static ushort Min(ushort val1, ushort val2);
public static double Pow(double x, double y);
public static decimal Round(decimal d);
public static double Round(double a);
public static decimal Round(decimal d, int decimals);
public static decimal Round(decimal d, MidpointRounding mode);
public static double Round(double value, int digits);
public static double Round(double value, MidpointRounding mode);
public static decimal Round(decimal d, int decimals, MidpointRounding
public static double Round(double value, int digits, MidpointRounding mode); ro.
public static int Sign(decimal value);
public static int Sign(double value);
public static int Sign(float value);
public static int Sign(int value);
public static int Sign(long value);
public static int Sign(sbyte value);
public static int Sign(short value);
public static double Sin(double a);
public static double Sinh(double value);
public static double Sqrt(double d);
public static double Tan(double a);

```

```
public static double Tanh(double value);
public static decimal Truncate(decimal d);
public static double Truncate(double d);
```

## ShowMsg

Showmsg is MessageBox.Show static method renamed to SHOWMSG. It is directly imported from :NET library. Possible arguments are.

```
public static DialogResult Show([IWin32Window owner, ]string text[, string caption[, MessageBoxButtons buttons[, MessageBoxIcon icon[, MessageBoxDefaultButton defaultButton[, MessageBoxOptions options[, string helpFilePath[, HelpNavigator navigator[, object param]]]]]]]]];
```

## LikeDB

Like db allows to make Sql queries to almost any databases. There is two procedures for that. Refer to sample code SQLDEMO.BAS. With DBEXform refer DBEXlib replaces this library.

```
public void DBConnect(string connectionstring)
```

Connect to database. Parameter is valid connection string for oledb.

```
public SQLResult SQL(string query)
```

Execute database query. Result set will be a result of the function.

```
class SQLResult
```

```
{
```

```
public SQLResult(DataTable table)
```

Constructor.

```
public object this[int row, string column]
```

Return value of row number and column by name or number.  
Rows and Columns are numbered from base 1.

```
public DataRow this[int row]
```

Return whole datarow by number.

```
public DataColumn Column(int inx)
```

Return datacolumn by index of column. Columns are indexed from base 1.

```
public int Columns
```

Return count of columns.

`public int Rows`

Return number of columns.

`public DataTable Table;`

Direct access database object.

## ***DBEXlib***

There are couple procedures inside of Dbexform- application. They are only usable inside DBEXform component events.

`public SQLResult SQL(string query)`

Execute sql statement to current database. Same command than LikeDB.SQL but uses only current database.

`public void First()`

Move current record pointer to first in list

`public void Rewind()`

Move current record pointer to previous in list

`public void Forward()`

Move current record pointer to next in list

`public void Last()`

Move current record pointer to last in list

`public void Post()`

Apply changes in record to database

`public void Cancel()`

Cancel all changes

`public void Delete()`

Delete record from database

`public void Insert()`

Start inserting new record

`public void Show(string ShowForm)`

Show modal form. Form fields could be modified after shown.

`public object` Result()

Retrieve DialogResult after Show.

`public object` Field

Assign values of fields into the form opened by Show command.

`public object` DataSet

Return current dataset in parent form or subform.

## **LikeIO**

`public static void` Run(`string` cmdLine)

Execute external command.

`public static bool` SaveFile(`string` filename, `object` data)

Save data to File named filename using UTF-8 encoding.

`public static bool` SaveFile(`string` filename, `object` data, `string` encode)

Save data to File named filename using encoding

`public static string[]` LoadFile(`string` filename)

Load text from given file named filename using UTF-8 encoding.

`public static string[]` LoadFile(`string` filename, `string` encode)

Load Source.text from given file named filename. Possible encodings are ASCII,UTF8 and UNICODE.

## **LikeBasicConsole**

These variables are only usable inside of LikeBasic console.

### **CAPTION**

Caption of LikeBasicConsole

### **PARENTFORM**

LikeBasicConsole self.Access windows Caption variable using syntax:

PARENTFORM.CAPTION

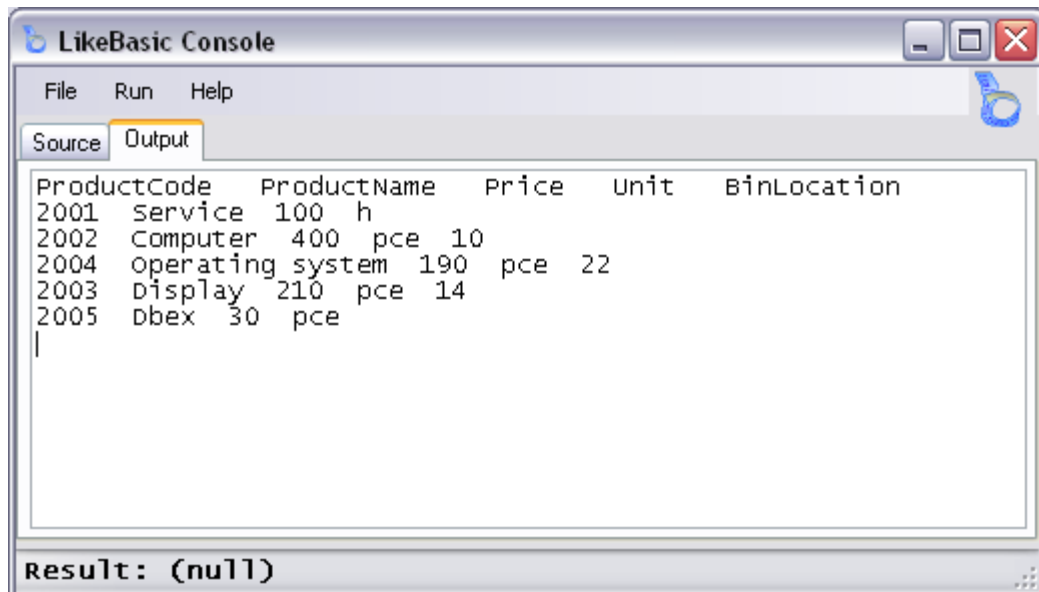
# 17. Samples

## SQLDemo

SqlDemo demonstrates how to make SQL queries to database and then print then result of dataset.

```
'fix Access file path if needed
DBConnect("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Program...
Files\LikeBasic\dbexSample1.mdb");
r=sql("select * from product") :other tables: customer,productorder

for i=1 to r.columns
  ? r.column(i) ; " ";
next i
?
for i=1 to r.rows
  for j=1 to r.columns
    ? r(i,j);" ";
  next j
  ?
next i
```



Result of sqldemo

## 8Queens

8 Queens is a problem for placing them to the chessboard so none of them could see each others. Program will find out all possible positions and list them on output text as chess coordinates.

It demonstrates how to use Recursive programming. For loop variables are always local. So there is no need to use dim command.

Source:

```

rem 8 Queens on chess board
cnt=1
AddQueen(1)

function AddQueen(queen) : ' rem add new Queen to chess bard

  for i=1 to 8

    queens(queen)=i
    if TestQueen(queen,i) then
      if queen=8 then
        ?cnt;"";:' Print queens
        cnt=cnt+1
        for j=1 to 8
          ? chr(64+Queens(j));j;" ";
        next j
        ?
      else
        AddQueen(queen+1)
      end if
    end if
  next i

function TestQueen(queen,pos) : ' rem test position if it is available to given queen
number

  for i=1 to queen-1
    dif=queen-i
    if queens(i)=pos or queens(i)-dif=pos or queens(i)+dif=pos then
      return 0
    end
  next i

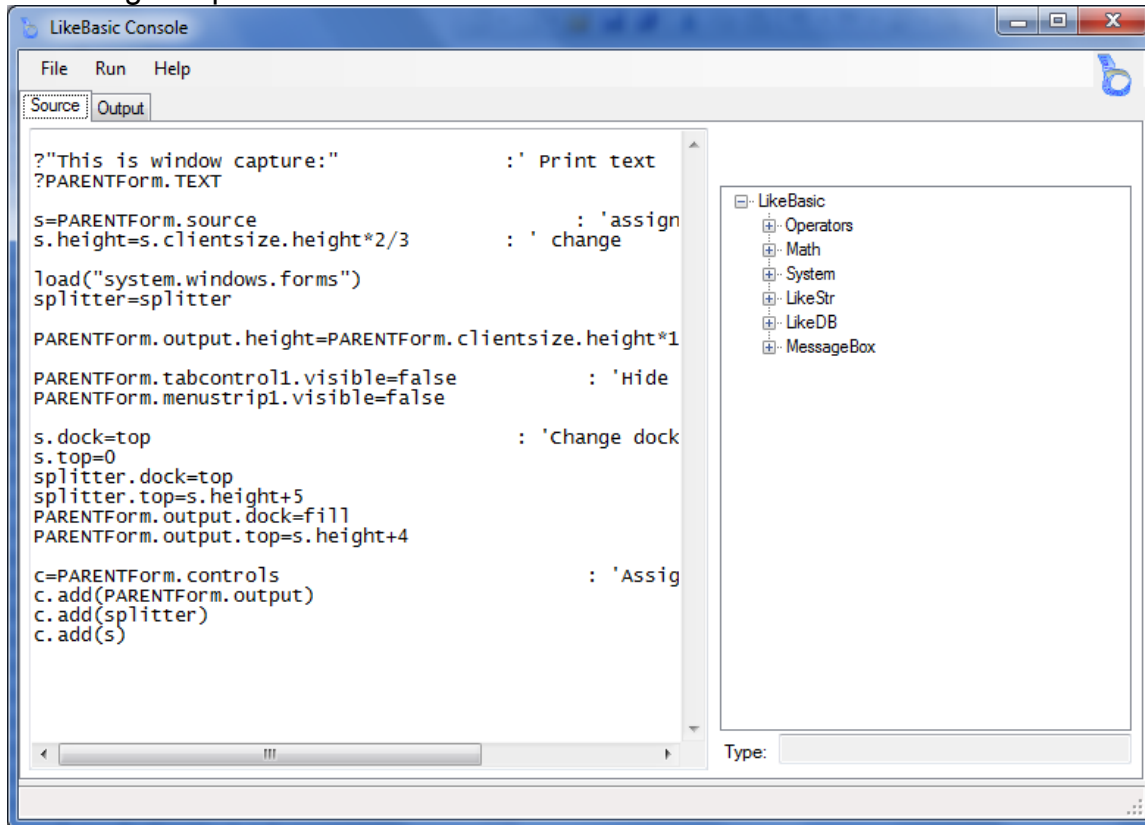
  return 1

```

## WINCUSTOM

Wincustom demonstrates how it is possible to modify existing window form with LikeBasic. Only component which have public modifier could be modified.

Following sample show console screen before and after modification.



```

LikeBasic Console

?"This is window capture:"           :' Print text
?PARENTForm.TEXT

s=PARENTForm.source                  : 'assign source component to s
s.height=s.clientsize.height*2/3    : ' change

load("system.windows.forms")
splitter=splitter

PARENTForm.output.height=PARENTForm.clientsize.height*1/3

PARENTForm.tabcontrol1.visible=false : 'Hide Controls
PARENTForm.menustrip1.visible=false

s.dock=top                           : 'change dock properties
s.top=0
splitter.dock=top

This is window capture:
LikeBasic Console

Result: Scripts.SSCRIPT

```

## Draw Sin Graph

Draw Sin demonstrates how to construct and use .Net Framework object. It also demonstrates use of events. Paint event is used to draw sin graph on the window.

Example:

*'Run without debugger to get maximum performance*

*LOAD("system.windows.forms") : 'Load system.windows.forms.dll from FW system folder*

*LOAD("system.drawing") : 'Load system.drawing.dll form FW system folder*

*pi=3.14*

*frm=Form : 'Create new instance of Form class*

*frm.Paint=event(frmPaint) : 'Assign Paint event handler*

*frm.size=size(700,300)*

*frm.text="DRAW SIN FUNCTION USING PAINT EVENT"*

*frm.show : 'Show created form*

*function frmPaint(sender, e)*

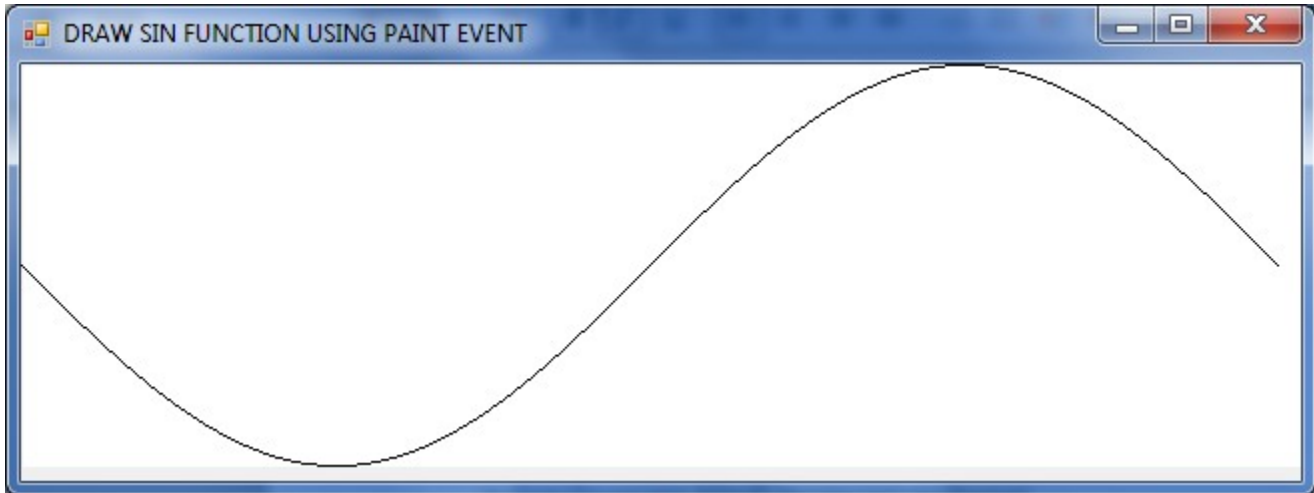
*' showmsg(paint)*

*g=e.graphics*

```

g.FillRectangle(brushes.white,1,1,640,200)
startpoint=point(0,100)
for i=0 to pi*2 step 0.01
    ? i*100,sin(i)*100+100
    endpoint=point(i*100,sin(i)*100+100)
    g.drawLine(pens.black,startpoint,endpoint)
    startpoint=endpoint
next I

```



Result of script

## Compress and decompress file

Comperess demo show how to work with streams. Script lets user to select file to compress and decompress using file dialog. Demonstration uses GzipStream class as compression routine. Compressed file will be stored in \*.gz and decompressed file name will be \*.gz.TXT.

Example

```

'Compress file using gz stream
LOAD("system.windows.forms") : 'Load system.windows.forms.dll from FW system folder

LOAD("system") : 'System.dll contains namespace system.io.compression
LOAD("mscorlib") : 'mscorlib contains namespace system.IO

fo=OpenFileDialog
if fo.showdialog=1 then
'Comperess File
instream=file.openread(fo.filename)
cname=Path.ChangeExtension(fo.filename, ".gz")

```

```

outstream=gzipstream(file.openwrite(cname),compressionmode.compress,0)
' ? outstream.gettype
copystream(instream,outstream)
instream.close
outstream.close

```

```

'DeCompress file
instream=gzipstream(file.openread(cname),compressionmode.decompress,0)
outstream=file.openwrite(cname;".TXT")
' ? outstream.gettype
copystream(instream,outstream)
instream.close
outstream.close
end if
?Done

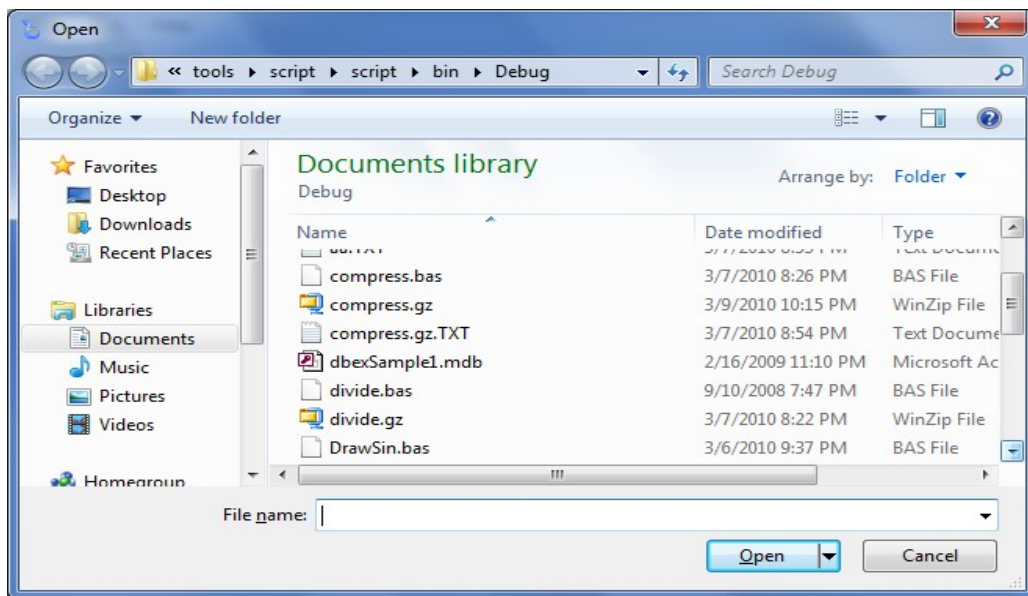
```

```
function copystream(fromstream,tostream)
```

```

buffer=Array.CREATEINSTANCE(Byte,1024)
n=1
while(n>0)
N=fromstream.READ(BUFFER,0,1024)
tostream.write(Buffer,0,n)
wend

```



Select file to compress, Compressed file has extension .gz. Decompressed gz.TXT

## Evaluate value

Evaluate value could be done simply by adding RETURN to the begin of the line. This sample could be tested with console. Result will be shown at bottom of the output.

```
RETURN caption; "*****"
```

Or shortened

```
= caption;"*****"
```

If there is possible to have multi line source, then it is possible to use functions to calculate value.

```
Return myfunc(caption)
```

```
Function myfunc(cap)
```

```
While len(cap)<30
```

```
  Cap=cap; "*" "
```

```
wend
```

```
Return CAP
```

## 18. Known issues

1. Debugging the code may lead to slow execution. So running for example 8Queens script in debugger, it will be executed very slowly.  
Solution: Don't use recursive scripts in debug environment.
2. Selecting right type in conversion when calling .NET when there are overloaded versions of function with different type of parameters.  
Solution: LB2.0 have new parameter ranking functionality to solve these problems.

## 19. Support

Web Page:

<http://www.willmansoft.com>

Email

[support@willmansoft.com](mailto:support@willmansoft.com)