



## Programming Language Reference Revision 0001, for BASIC release Alpha 021

### Data Types

The language defines three intrinsic (primitive) data types. These data types are string, integer and float. All literal values are of one of these types.

A literal value is one that you write as it is in the source code. For example the value 100 or the word "Jose".

### Strings

A literal string value is a group of characters, a word or a series of words. A literal string value is always written inside double quotes. The following representations are all valid string literals:

```
"Jose L. Cuevas"      "HELLO WORLD"      "^%&$^%#%$HGVFC"
```

String literals also support escape sequences like "\n". An escape sequence allows you to change the meaning of a character in a string. An escape sequence starts with the character "\" followed by a single character. When an escape sequence is found inside a string literal it is replaced by its meaning.

Here is a list of escape sequences supported by BASIC:

**\n** inserts a new line character. In Mac is ASCII 13 in Windows is ASCII 13 and ASCII 10.

**\t** inserts a tab (eight spaces).

**\r** inserts ASCII 10.

**\f** inserts ASCII 13.

**\"** inserts a double quote inside a string.

Escape sequences allow to do things like:

```
"Hey \"This is cool\"!!!!"
```

Notice that a string literal starts and ends with a double quote, without the escape sequence the above string literal would generate an error since it is not valid. The above string literal becomes:

```
Hey "This is cool"!!!!
```

For the rest of the documentation any string literal will be called a **STRING**.

## Numbers

In source code you can write numeric literals. A numeric literal may be any numbers, it may start with a negative symbol. Decimal portions must be prefixed with a zero if no whole part is included. For example:

23      1.07              -5              14567              0.5      0.3e+10

A number without a decimal portion is known in BAXIC as an **INTEGER**. A number with a decimal portion is a **FLOAT**.

### INTEGER:

An integer is a whole number. Its range is from +- **2,147,438,648**. Use the math library function `cint` to cast a value to an integer. An integer in BAXIC is considered a **LONG** in other languages.

### FLOAT:

A **FLOAT** is a number that may contain a decimal part. Its range is from **2.2250738585072013 e-308** and **1.7976931348623157 e+308**. Use the math library function `cfloat` to cast a value to a float. A float is also known as a double precision real number.

Currently BAXIC provides little support for very large floating point/real numbers, yet current support should be enough for most calculations. We hope that in a future support is added for advanced scientific and mathematical calculations.

## Variables

Variables like in any other programming language allow you to store and represent a value (a data type). A variable can store one value of a given data type. The data type of a variable determines what type of data the variable can store.

A variable can be of type **STRING**, **INTEGER**, **FLOAT** or any other data type supported by the language.

Variables are represented by a name. Each variable must have a unique name in a given scope. The name of variables must start with an alphabetic character and may NOT contain spaces or punctuation symbols other than the underscore. The name of a variable may not be equal to a reserved keyword or the name of an object or used defined structure or class.

To declare a variable use the **DIM** statement. You have to declare a variable before you use it. The syntax of the **DIM** statements is:

```
DIM variablename as datatype
```

## User Defined Data Types

A User Defined Data Type or UDT is like a special data type that the user defines. Sometimes the basic data types are not enough or do not allow for an easy representation of certain data or relationship among the data. For example if you wanted to represent an employee in code you could use string variables like "Name", "Title", or a numeric variable like "Salary". Yet all of these variables have something in common and that is that they represent an employee. They have a common relationship. Not only that but a variable salary may apply for something else in your code, therefore you may want to know that this is the salary of an employee not some other salary. For this particular example it would be great if we could define a variable of type "employee" that had properties like "name", "salary" and others.

BASIC allows you to define two types of UDTs, Structures and Classes. The structure is the most basic of the two.

### Structures

The structure also known as a Type Structure allows you to group a series of variables under another variable. We define a type structure as follows:

```
Type Car
    Brand as string
    Model as string
    Year as integer
End Type
```

We use the keyword "TYPE" to indicate that we want to define a type structure. The keyword Type is followed by the name of our new data type. Inside our type block we define the properties or variables that made up our new data type. We finally close the type structure with the keywords "END TYPE".

In the above example we have defined a new data type named Car. This data type has the properties Brand, Model and Year. Now we can declare a variable of type Car as follows:

```
DIM MyCar as Car
```

With this new variable of type Car now we can say things like:.

```
MyCar.Model = "XTerra"
MyCar.Year = 2004
```

In the above examples you may have notice that we use the dot (period) to access a property of our type structure. Notice also that the name of the properties are that same as the ones we used to define our structure.

### Classes

A class just like a type structures allows you to group properties with a particular relationship under one data type. Classes have one advantage over structures and it is that they let you associate code that you may execute along with your properties. In the

example above lets said we add the property "EngineModel" to the structure. If you want to change the "EngineModel" you have to call a function or add the code to change the variable. Notice this code would be somewhere in your program and you must duplicate the code every time you want to do the same. With a class I can put the code together with my new data type. Lets see how we declare a class:

```
Class Car
    Public Property Brand as string
    Public Property Model as string
    Public Property Year as integer
    Public Property EngineModel as string

    Public Sub ChangeEngineModelToBasic( )
        me.EngineModel = "Basic"
    End Sub

End Class
```

The class is like a blueprint in order to use it we must declare a variable from our class.

```
Dim myCar as car
myCar = new car
```

The code above declares a variable of type car and creates a new instance of the class car by using the new operator. The variable mycar is known as an object while car is the class. Whenever we use the new operator it is said that we are creating a new instance. The variable mycar now has the properties defined in the class car like for example Brand and Model.

```
Dim mycar as car
mycar = new car

mycar.brand = "Dodge"
mycar.model = "Dakota"
mycar.ChangeEngineModelToBasic()
```

When you need a new class to have the same methods and variables from an existing class you can use the statement extends when declaring a new class. The extended class has all the methods and variables of the base class.

```
Class Car
    Public Property Brand as string
    Public Property Model as string
    Public Property Year as integer
    Public Property EngineModel as string

    Public Sub ChangeEngineModelToBasic( )
        me.EngineModel = "Basic"
    End Sub

End Class
```

```

Class truck
    extends car
    Public Property fourwheel as integer

End Class

```

All instances of objects inherit from a base class called Object. The Object class has no properties. The object class does defines a few handy methods:

```

object._name() as string    Returns the name of the object.

object._datatype() as string    Returns the data type of the object.

object._classname() as string    Returns the class name of the
object.

object._hasMember(name as string) as boolean Returns true if the
object has a member with the name given in the parameter name.

```

Here is an example of using these methods:

```

class ckNiceClass
    public property name as string
    public property count as integer

    public sub ckNiceClass()
        msgbox("Created new ckNiceClass")
    end sub
end class

sub main()
    dim test as ckNiceClass

    test = new ckNiceClass
    //we can get the data type
    msgbox("Test is a " & test._DataType())
    msgbox("Name is " & test._Name())
    //we can check if an object implements a member
    if test._hasMember("name") then
        test.name = "Jose L. Cuevas"
        msgbox("ok, I can use this property")
    else
        msgbox("upps, this object is a no-no...")
    end if
end sub

```

## Control Flow Statements

## IF STATEMENT

Conditionally executes a group of statements, depending on the value of a boolean expression.

```
IF expression THEN
    Statement...
[ELSEIF expression THEN
    Statement...]
[ELSE
    Statement...]
END IF
```

## FOR NEXT STATEMENT

Executes a series of statements a specified number of times.

```
FOR counter = start TO end [STEP step]
    Statement...
NEXT

FOR counter = start DOWNTO end
    Statement...
NEXT
```

Counter is an integer variable that will be incremented. Start can be any numeric expression or variable that returns an integer, this is the initial value. End is an integer variable or constant, this is the final value. Step is an integer variable or constant. When step is included counter will be incremented.

The statements will be executed each time until counter is greater than end or if DownTo is used until counter is less than end .

Another variation of the FOR...NEXT loop is the FOR...IN loop. The FOR...IN loop allows you to easily loop through the elements of an array or list. For more info see [arrays](#).

## DO...LOOP STATEMENT

Repeatedly executes a group of statements while the value of an expression is true.

```
DO UNTIL expression
    Statement...
LOOP

DO WHILE expression
    Statement...
LOOP
```

## SELECT CASE STATEMENT

Conditionally executes one group of statements from several groups, depending on the value of an expression.

```

SELECT CASE expression
        CASE value1[,value2]...
            Statement Group1...
        [CASE value3
            Statement Group2...]
        .
        .
        .
        [DEFAULT
            Statement Group...]
END SELECT

```

The expression must evaluate to a numeric or string value. The statement group inside the DEFAULT is executed if none of the previous CASE values match the expression.

## LIST AND ARRAYS

Lists are a form to group values under one variable. Lists are also known as Arrays or Matrix. To access a particular value in a List we use an index or a key value.

In BAXIC there are two types of Lists the traditional Array which we access by a numeric index and a Hash type which we access using the name of the value as the key.

## ARRAY

To define an array we use a DIM statement. The DIM statement looks like this:

```
DIM MyArray[5] AS STRING
```

In the example above the array is named `MyArray`. Inside the brackets we indicate the amount or rows in our array or in other words we set the dimensions. The first row in the array is row zero (0) and the last row is row five (5). Our array has a total of 6 rows. The brackets are followed by the data type of our array. In this example the array will store six rows of strings.

To access a given row in code we use the name of the array followed by the brackets. Inside the brackets we indicate the index of the row we want to manipulate. If we attempt to access a row past the last row defined an "Out of bounds" error will be raised. The same will happen if we try to access a negative row (lower than 0).

```

MyArray[0] = "Jose"
MyArray[1] = "Joe"
MyArray[2] = "Lourdes"
msgbox(MyArray[2])

```

```

Sub Main()
    dim s as string

    msgbox("Array and Array of objects")

    dim a(3) as string
    a(0) = "Jose"

```

```

a(1) = "L"
a(2) = "Cuevas"
s = a(0) & " " & a(1) & " " & " " & a(2)
msgbox("Hello " & s)

dim d(3) as date
d(0) = new date
d(1) = new date
d(2) = new date

d(2) = d(1)

if d(1).ParseDate("12/25/02") = true then
    msgbox("Shortdate: " & d(1).shortdate)
end if

msgbox("Long date d(0): " & d(0).longdate)
msgbox("Long date d(1): " & d(1).longdate)
msgbox("Long date d(2): " & d(2).longdate)
end sub

```

## PACKING VALUES INTO A LIST

The LIST function allows you to pack values into a list. This function is very useful when you want to return multiple values or manipulate multiple values in a FOR-LOOP.

The syntax for the LIST function is as follows:

```
LIST( value1, value2, value3, valueN... ) as List
```

The list function takes N number of values each separated by a comma. A value can be a variable or a literal value like a number or a string. List will return an array of type LIST. If one of the arguments is a variable a reference to the actual variable is used instead of a copy.

```

Sub Main()
    DIM i as string
    DIM r as list

    r = getMultipleValues()
    msgbox(r["name"] & " " & r["lname"])
    msgbox("The age is " & r["age"])
End Sub

Function getMultipleValues()
    DIM name as string, lname as string
    DIM age as integer

    name = "Jose"
    lname = "Cuevas"
    age = 30

```

```

        Return list(name,lname,age)
End Function

Sub Main()
    DIM name AS string,lname AS string
    DIM age AS integer
    list(name,lname,age) = getMultipleValues()
    msgbox(name & " " & lname & " of age " & age)
End Sub

Function getMultipleValues()
    DIM name AS string, lname AS string
    DIM age AS integer
    name = "Jose"
    lname = "Cuevas"
    age = 30

    Return list(name,lname,age)
End Function

```

A variable of type LIST has can also be treated as an object. All variables of type LIST implement the following methods:

```

list.empty()      Remove all elements of the list.

list.contains(key as string) as boolean      Returns true if an
element with name given by key exists.

list.remove(key as string) Removes the element with name given in
key.

list.removeLast()      Removes the last element.

list.removeFirst()      Removes the first element.

list.count() as integer      Returns the number of elements.

list.lastIndex() as integer Returns the index of the last element.

list.asString([delimiter as string]) as string Return all
elements concatenated using delimiter or a space if delimiter is not provided.

```

An example of using the LIST methods:

```

sub main()
    dim aList as collection
    dim a as string, b as string
    a = "jose"
    b = "cuevas"

```

```

aList = list(a, b)

aList["a"] = "Joe"
msgbox("count=" & aList.count())
msgbox(aList["a"] & " " & aList["b"])

end sub

```

## PACKING NUMERIC VALUES USING RANGE

The RANGE function allows you to create a list of numeric values. The syntax for the RANGE function is as follows:

```
RANGE( StartValue, EndValue ) as List
```

The RANGE function takes two numeric arguments. The first is the integer value where the list will start and the second is the value where the list will end. The list returned by range will have EndValue minus StartValue rows. The first row of the list will have a value of StartValue the second will have a value of StartValue plus one and so on until EndValue.

```

Sub Main()
    DIM n AS string
    FOR n in list("jose","joe","lourdes","cuevas")
        msgbox(n)
    NEXT

    msgbox("Exit with value: " & n)

    DIM i AS integer

    FOR i IN list(5,6,7,8,9,10)
        msgbox("Value of i: " & i)
    NEXT

    msgbox("Testing range")
    FOR i IN range(30,45)
        msgbox("Value of i: " & i)
    NEXT
End Sub

```

## User declared Functions

Methods allow you to break up your code in reusable segments that can be called from your code. There are two types of methods in BAXIC a SUB and a FUNCTION. A SUB is a

method that does something and does not return any value. The `FUNCTION` in the other hand is just like a `SUB` but it returns a value.

### Declaring methods that return values

We use the `FUNCTION` statement to declare a method that can be used on the right side of an expression. You can declare a `FUNCTION` globally or inside a class. The `FUNCTION` statements allows you to name the method, define its parameters, set the data type of the returned value and the code to be executed when the method is called. The syntax is:

```
function name([p1 as type[, p2 as type][,...]]) as type
    [local variables...]
    [statements...]
    return value
end function
```

The name of the function must follow the same naming conventions of variables. You will call the function in your code using this name. Inside the parenthesis you must list the parameters required to call your function. Commas separate multiple parameters. A parameter is declared just like variables but without the `DIM` statement. Finally the data type of the returned value is indicated.

A `FUNCTION` or `SUB` cannot be defined inside another `SUB` or `FUNCTION`. When the `FUNCTION` is called the code inside the `FUNCTION` statement will be executed from top to bottom. The execution of your code will return to the line that called the function once the code in the function is executed.

You call a `FUNCTION` by its name followed by the parameters inside the parentheses.

A method is like a tiny sub program and as such you can define variables inside a method just like you would. Variables declared with the `DIM` statement inside the method are considered local in scope and they only exist inside the method.

To declare functions we use the `FUNCTION` statement in BAXIC. A

## Built In Classes

A description of classes built-in BAXIC.

### Class Object

**Super** Object

**Properties:**

**Methods:**

- \_name(obj as pointer) as string
- \_datatype(obj as pointer) as string
- \_classname(obj as pointer) as string
- \_hasMember(obj as pointer, value as string) as integer

### Class Math

**Super** Object

**Properties:**

**Methods:**

- rnd() as float
- abs(obj as pointer, value as float) as float
- acos(obj as pointer, value as float) as float
- asin(obj as pointer, value as float) as float
- atan(obj as pointer, value as float) as float
- atan2(obj as pointer, value as float) as float
- bin(obj as pointer, value as float) as string
- oct(obj as pointer, value as float) as string
- hex(obj as pointer, value as float) as string
- ceil(obj as pointer, value as float) as integer
- exp(obj as pointer, value as float) as float
- floor(obj as pointer, value as float) as integer
- log(obj as pointer, value as float) as float
- random(obj as pointer, rangeStart as integer,rangeEnd as integer) as integer
- max(obj as pointer, value1 as float,value2 as float) as float
- min(obj as pointer, value1 as float,value2 as float) as float
- pow(obj as pointer, value1 as float,value2 as float) as float
- mod(obj as pointer, value1 as float,value2 as float) as float
- round(obj as pointer, value as float) as float
- sqrt(obj as pointer, value as float) as float
- sin(obj as pointer, value as float) as float
- tan(obj as pointer, value as float) as float
- cos(obj as pointer, value as float) as float
- val(obj as pointer, value as string) as float
- cInt(obj as pointer, value as float) as integer
- cFloat(obj as pointer, value as integer) as float
- format(obj as pointer,value as float,format as string) as string
- char(obj as pointer,[value] as integer) as string
- chr(obj as pointer,[value] as integer) as string

### Class Strlib

**Super** Object

## Properties:

## Methods:

chrAtB([value] as integer) as string  
chrAt([value] as integer) as string  
lcase([value] as string) as string  
ucase([value] as string) as string  
split([source] as string,delimiter as string) as string  
str(value as float) as string  
asc([value] as string) as integer  
len([value] as string) as integer  
trim([value] as string) as string  
ltrim([value] as string) as string  
rtrim([value] as string) as string  
soundex([value] as string) as string  
instr([start] as integer,source as string, find as string) as integer  
nthfield([source] as string,del as string,index as integer) as string  
countfield([source] as string,del as string) as integer  
strComp([str1] as string,str2 as string,[mode] as string) as integer  
left([value] as string,count as integer) as string  
right([value] as string,count as integer) as string  
mid([source] as string,start as integer,[length] as integer) as string  
replace([source] as string,find as string,replace as string) as string  
replaceall([source] as string,find as string,replace as string) as string

## Class Date

### Super Object

## Properties:

date.Nil as integer  
date.Hour as integer  
date.Minute as integer  
date.Day as integer  
date.Month as integer  
date.Second as integer  
date.Year as integer  
date.WeekofYear as integer  
date.DayofWeek as integer  
date.DayofYear as integer  
date.TotalSeconds as float  
date.ShortDate as string  
date.LongDate as string  
date.LongTime as string  
date.ShortTime as string  
date.LeapYear as integer  
date.AbbreviatedDate as string

## Methods:

FormatDate(obj as pointer, s as string) as string  
CookieTime(obj as pointer) as string  
ParseDate(obj as pointer, s as string) as integer  
GotoToday(obj as pointer)

## Class Memoryblock

**Super Object**

### Properties:

memoryblock.LittleEndian as integer  
memoryblock.size as integer

### Methods:

SetBooleanValue(obj as pointer,offset as integer,value as integer)  
SetByteValue(obj as pointer,offset as integer,value as integer)  
SetDoubleValue(obj as pointer,offset as integer,value as integer)  
SetSingleValue(obj as pointer,offset as integer,value as integer)  
SetLongValue(obj as pointer,offset as integer,value as integer)  
SetShortValue(obj as pointer,offset as integer,value as integer)  
SetUShortValue(obj as pointer,offset as integer,value as integer)  
SetStringValue(obj as pointer,offset as integer, length as integer, value as string)  
SetPString(obj as pointer,offset as integer, value as string)  
SetCString(obj as pointer,offset as integer, value as string)  
GetBooleanValue(obj as pointer,offset as integer) as integer  
GetByteValue(obj as pointer,offset as integer) as integer  
GetDoubleValue(obj as pointer,offset as integer) as float  
GetSingleValue(obj as pointer,offset as integer) as float  
GetLongValue(obj as pointer,offset as integer) as integer  
GetShortValue(obj as pointer,offset as integer) as integer  
GetUShortValue(obj as pointer,offset as integer) as integer  
GetStringValue(obj as pointer,offset as integer, length as integer) as string  
GetPString(obj as pointer,offset as integer) as string  
GetCString(obj as pointer,offset as integer) as string  
LeftB(obj as pointer,length as integer) as object  
RightB(obj as pointer,length as integer) as object  
MidB(obj as pointer,offset as integer,length as integer) as object

## Class File

**Super Object**

### Properties:

file.Nil as integer  
file.count as integer  
file.Directory as integer  
file.Exists as integer  
file.Alias as integer  
file.Length as float  
file.Locked as integer  
file.MacDirID as integer  
file.IsReadable as integer  
file.IsWritable as integer  
file.Visible as integer  
file.AbsolutePath as string  
file.MacCreator as string  
file.MacType as string  
file.name as string  
file.ModificationDate as date

file.CreationDate as date  
file.Parent as file

**Methods:**

CopyFileto(obj as pointer, f as file)  
MoveFileto(obj as pointer, f as file)  
CreateAsFolder(obj as pointer)  
Delete(obj as pointer)  
Launch(obj as pointer)  
Close(obj as pointer)  
OpenAsTextFile(obj as pointer) as object  
CreateTextFile(obj as pointer) as object  
OpenAsSound(obj as pointer) as object  
OpenAsPicture(obj as pointer) as object  
OpenAsPNG(obj as pointer) as object  
Item(obj as pointer, index as integer) as object  
Child(obj as pointer, name as string) as object

**Class Color**

**Super** Object

**Properties:**

color.red as integer  
color.green as integer  
color.blue as integer  
color.hue as float  
color.saturation as float  
color.value as float  
color.cyan as float  
color.magenta as float  
color.yellow as float

**Methods:**

gethexcolor(obj as pointer) as string  
createfromhex(obj as pointer, value as string)  
selectcolor(obj as pointer, prompt as string)  
rgb(obj as pointer, r as integer, g as integer, b as integer)  
hsv(obj as pointer, h as float, s as float, v as float)  
cmy(obj as pointer, c as float, m as float, y as float)

**Class Regex**

**Super** Object

**Properties:**

Regex.Nil as integer  
Regex.searchpattern as string  
Regex.replacementPattern as string  
Regex.casesensitive as integer  
Regex.greedy as integer  
Regex.dotmatchall as integer

**Methods:**

replace(obj as pointer, source as string, [startat] as integer) as object

search(obj as pointer,source as string,[startat] as integer) as object

### Class Regexpmatch

**Super** Object

**Properties:**

regExMatch.Nil as integer  
regExMatch.count as integer

**Methods:**

subExpressionString(obj as pointer,index as integer) as string  
subExpressionStart(obj as pointer,index as integer) as integer

### Class Tcpsocket

**Super** Object

**Properties:**

tcpSocket.Port as integer  
tcpSocket.Address as string  
tcpSocket.RemoteAddress as string  
tcpSocket.BytesAvailable as integer  
tcpSocket.BytesLeftToSend as integer  
tcpSocket.IsConnected as integer  
tcpSocket.LastErrorCode as integer  
tcpSocket.LocalAddress as integer

**Methods:**

Read(obj as pointer,bytes as integer) as string  
ReadAll(obj as pointer) as string  
lookahead(obj as pointer) as string  
Write(obj as pointer,Data as string)  
Close(obj as pointer)  
Listen(obj as pointer)  
Connect(obj as pointer)  
Disconnect(obj as pointer)  
Pull(obj as pointer)  
Purge(obj as pointer)

### Class Textinputstream

**Super** Object

**Properties:**

TextInputStream.Nil as integer  
TextInputStream.EOF as integer

**Methods:**

Close(obj as pointer)  
ReadAll(obj as pointer) as string  
ReadLine(obj as pointer) as string

### Class Textoutputstream

**Super** Object

**Properties:**

TextOutputStream.Nil as integer

TextOutputStream.Delimiter as string

**Methods:**

Close(obj as pointer)  
Write(obj as pointer,s as string)  
WriteLine(obj as pointer,s as string)

**Class Timer**

**Super** Object

**Properties:**

timer.Nil as integer  
timer.Mode as integer  
timer.Period as integer

**Methods:**

eventhandler(obj as pointer,eventName as string, fp as object)  
reset(obj as pointer)

**Class System**

**Super** Object

**Properties:**

System.Nil as integer  
System.UserCancelled as integer  
System.Desktop as file  
System.Preferences as file  
System.Temporary as file  
System.Ticks as float  
System.Microseconds as float  
System.VolumeCount as integer  
System.ScreenCount as integer

**Methods:**

DoEvents(obj as pointer)  
Yield(obj as pointer)  
SetEnv(obj as pointer, key as string,value as string)  
GetEnv(obj as pointer, key as string) as string  
Volume(obj as pointer,index as integer) as object  
Screen(obj as pointer, index as integer) as object

**Class Keyboard**

**Super** Object

**Properties:**

Keyboard.Nil as integer  
Keyboard.AsyncCommandKey as integer  
Keyboard.AsyncControlKey as integer  
Keyboard.AsyncOptionKey as integer  
Keyboard.AsyncShiftKey as integer  
Keyboard.ControlKey as integer  
Keyboard.OptionKey as integer  
Keyboard.ShiftKey as integer

**Methods:**

AsyncKeyDown(keycode as integer) as integer  
Keyname(keycode as integer) as string

**Class Screen****Super Object****Properties:**

screen.Nil as integer  
screen.Depth as integer  
screen.Height as integer  
screen.Width as integer  
screen.Left as integer  
screen.Top as integer  
screen.AvailableHeight as integer  
screen.AvailableWidth as integer  
screen.AvailableLeft as integer  
screen.AvailableTop as integer

**Methods:****Class Graphics****Super Object****Properties:**

graphics.Nil as integer  
graphics.left as integer  
graphics.top as integer  
graphics.height as integer  
graphics.width as integer  
graphics.visible as integer  
graphics.enabled as integer  
graphics.font as string  
graphics.fontsize as integer  
graphics.fontbold as integer  
graphics.fontunderline as integer  
graphics.fontitalic as integer  
graphics.forecolor as color  
graphics.penheight as integer  
graphics.penwidth as integer  
graphics.textascent as integer  
graphics.TextHeight as integer  
graphics.copies as integer  
graphics.Firstpage as integer  
graphics.Lastpage as integer

**Methods:**

DrawLine(obj as pointer,x1 as integer,y1 as integer,x2 as integer,y2 as integer)  
DrawRect(obj as pointer,x as integer,y as integer,w as integer,h as integer)  
FillRect(obj as pointer,x as integer,y as integer,w as integer,h as integer)  
DrawOval(obj as pointer,x as integer,y as integer,w as integer,h as integer)  
FillOval(obj as pointer,x as integer,y as integer,w as integer,h as integer)  
DrawRoundRect(obj as pointer,x as integer,y as integer,w as integer,h as integer,r as

integer)

FillRoundRect(obj as pointer,x as integer,y as integer,w as integer,h as integer,r as integer)

DrawString(obj as pointer,s as string,x as integer,y as integer,[w] as integer)

SetPixel(obj as pointer,c as string,x as integer,y as integer)

DrawPicture(obj as pointer,p as picture,x1 as integer,y1 as integer,[w1] as integer,[h1] as integer,[x2] as integer,[y2] as integer,[w2] as integer,[h2] as integer)

GetPixel(obj as pointer,x as integer,y as integer) as string

StringWidth(obj as pointer,s as string) as integer

Refresh()

## Class Picture

**Super** Object

### Properties:

picture.height as integer

picture.width as integer

picture.graphics as graphics

picture.Mask as picture

picture.ImageCount as integer

picture.HorizontalResolution as integer

picture.VerticalResolution as integer

picture.depth as integer

picture.Transparent as integer

### Methods:

eventhandler(obj as pointer,eventName as string, fp as object)

IndexedImage(obj as pointer,index as integer) as object

## Class Sound

**Super** Object

### Properties:

sound.Nil as integer

### Methods:

IsPlaying(obj as pointer) as integer

Play(obj as pointer)

PlayLooping(obj as pointer)

Stop(obj as pointer)

Volume(obj as pointer,value as integer)

## Class Window

**Super** Object

### Properties:

window.title as string

window.left as integer

window.top as integer

window.height as integer

window.width as integer

window.visible as integer

window.picture as picture

window.backcolor as string

**Methods:**

eventhandler(eventName as string, fp as object)  
CreateControl(obj as pointer, ctrl as pointer, x as integer, y as integer, w as integer, h as integer,)  
Close(obj as pointer)  
Hide(obj as pointer)  
UpdateNow(obj as pointer)  
Show(obj as pointer)  
ShowModal(obj as pointer)  
Move(obj as pointer,x as integer,y as integer)  
Refresh(obj as pointer)  
SetFocus(obj as pointer)  
SetShapeFromImage(obj as pointer, image as picture, mask as picture)  
SetTransparency(obj as pointer, alpha as float)

**Class Button****Super Object****Properties:**

button.active as integer  
button.enabled as integer  
button.helptag as string  
button.left as integer  
button.top as integer  
button.height as integer  
button.width as integer  
button.visible as integer  
button.bold as integer  
button.italic as integer  
button.Font as string  
button.underline as integer  
button.FontSize as integer

**Methods:**

SetFocus(obj as pointer)  
Refresh(obj as pointer)  
RefreshRect(obj as pointer, x as integer, y as integer, width as integer, height as integer)  
Push(obj as pointer)

**Class Textbox****Super Object****Properties:**

textbox.active as integer  
textbox.enabled as integer  
textbox.helptag as string  
textbox.left as integer  
textbox.top as integer  
textbox.height as integer  
textbox.width as integer  
textbox.visible as integer

textbox.bold as integer  
textbox.italic as integer  
textbox.Font as string  
textbox.underline as integer  
textbox.FontSize as integer  
textbox.Text as string  
textbox.TextColor as color  
textbox.Format as string  
textbox.Mask as string  
textbox.AcceptTabs as integer  
textbox.ReadOnly as integer  
textbox.ScrollPosition as integer  
textbox.SelBold as integer  
textbox.SelItalic as integer  
textbox.SelUnderline as integer  
textbox.SelStart as integer  
textbox.SelLenght as integer  
textbox.SelFont as string  
textbox.SelText as string  
textbox.SelFontSize as integer  
textbox.UseFocusRing as integer  
textbox.SelColor as color

#### **Methods:**

SetFocus(obj as pointer)  
Refresh(obj as pointer)  
RefreshRect(obj as pointer, x as integer, y as integer, width as integer, height as integer)  
Copy(obj as pointer)  
Paste(obj as pointer)  
charPosAtLineNum(obj as pointer,value as integer) as integer  
charPosAtXY(obj as pointer,x as integer, y as integer) as integer  
lineNumAtCharPos(obj as pointer,value as integer) as integer

#### **Class Textarea**

**Super** Object

#### **Properties:**

textarea.active as integer  
textarea.enabled as integer  
textarea.helptag as string  
textarea.left as integer  
textarea.top as integer  
textarea.height as integer  
textarea.width as integer  
textarea.visible as integer  
textarea.bold as integer  
textarea.italic as integer  
textarea.Font as string  
textarea.underline as integer  
textarea.FontSize as integer  
textarea.Text as string

textarea.TextColor as color  
textarea.Format as string  
textarea.Mask as string  
textarea.AcceptTabs as integer  
textarea.ReadOnly as integer  
textarea.ScrollPosition as integer  
textarea.SelBold as integer  
textarea.SelItalic as integer  
textarea.SelUnderline as integer  
textarea.SelStart as integer  
textarea.SelLenght as integer  
textarea.SelFont as string  
textarea.SelText as string  
textarea.SelFontSize as integer  
textarea.UseFocusRing as integer  
textarea.SelColor as color

**Methods:**

SetFocus(obj as pointer)  
Refresh(obj as pointer)  
RefreshRect(obj as pointer, x as integer, y as integer, width as integer, height as integer)  
Copy(obj as pointer)  
Paste(obj as pointer)  
charPosAtLineNum(obj as pointer,value as integer) as integer  
charPosAtXY(obj as pointer,x as integer, y as integer) as integer  
lineNumAtCharPos(obj as pointer,value as integer) as integer

**Class Progressbar**

**Super** Object

**Properties:**

ProgressBar.active as integer  
ProgressBar.enabled as integer  
ProgressBar.helptag as string  
ProgressBar.left as integer  
ProgressBar.top as integer  
ProgressBar.height as integer  
ProgressBar.width as integer  
ProgressBar.visible as integer  
ProgressBar.Maximum as integer  
ProgressBar.Value as integer

**Methods:**

SetFocus(obj as pointer)  
Refresh(obj as pointer)  
RefreshRect(obj as pointer, x as integer, y as integer, width as integer, height as integer)

**Class Canvas**

**Super** Object

**Properties:**

canvas.active as integer  
canvas.enabled as integer  
canvas.helptag as string  
canvas.left as integer  
canvas.top as integer  
canvas.height as integer  
canvas.width as integer  
canvas.visible as integer  
canvas.picture as picture  
canvas.graphics as graphics

**Methods:**

SetFocus(obj as pointer)  
Refresh(obj as pointer)  
RefreshRect(obj as pointer, x as integer, y as integer, width as integer, height as integer)  
scroll(obj as pointer, deltax, deltay, [left] as integer, [top] as integer, [width] as integer, [height] as integer, [scrollcontrols] as integer)

**Class Checkbox**

**Super Object**

**Properties:**

checkbox.active as integer  
checkbox.enabled as integer  
checkbox.helptag as string  
checkbox.left as integer  
checkbox.top as integer  
checkbox.height as integer  
checkbox.width as integer  
checkbox.visible as integer  
checkbox.bold as integer  
checkbox.italic as integer  
checkbox.Font as string  
checkbox.underline as integer  
checkbox.FontSize as integer  
checkbox.caption as string  
checkbox.Value as integer

**Methods:**

SetFocus(obj as pointer)  
Refresh(obj as pointer)  
RefreshRect(obj as pointer, x as integer, y as integer, width as integer, height as integer)

**Class Mime**

**Super Object**

**Properties:**

MIME.Nil as integer

**Methods:**

Base64Decode(obj as pointer, value as string) as string

Base64Encode(obj as pointer, value as string) as string

### Class Http

**Super** Object

**Properties:**

http.Nil as integer

**Methods:**

URLDecode(obj as pointer, value as string) as string

URLEncode(obj as pointer, value as string) as string

### Class Url

**Super** Object

**Properties:**

url.Nil as integer

url.url as string

url.protocol as string

url.host as string

url.port as string

url.path as string

url.queryString as string

**Methods:**

Rebuild(obj as pointer)

### Class Mysqldbatabase

**Super** Object

**Properties:**

mysqldbatabase.Nil as integer

mysqldbatabase.databaseName as string

mysqldbatabase.timeout as integer

mysqldbatabase.host as string

mysqldbatabase.port as integer

mysqldbatabase.username as string

mysqldbatabase.password as string

mysqldbatabase.ErrorMessage as string

mysqldbatabase.Error as integer

mysqldbatabase.ErrorCode as integer

**Methods:**

Close()

Commit()

Rollback()

connect() as integer

runQuery(sql as string)

getRecordset(sql as string) as object

getInsertID() as integer

getAffectedRecords() as integer

### Class Recordset

**Super** Object

**Properties:**

recordset.Nil as integer  
recordset.bof as integer  
recordset.eof as integer  
recordset.fieldCount as integer  
recordset.recordCount as integer

**Methods:**

SetBooleanField(name as string, value as integer)  
SetIntegerField(name as string, value as integer)  
SetStringField(name as string, value as string)  
SetDoubleField(name as string, value as float)  
SetField(name as string, value as string)  
getBooleanField(name as string) as integer  
getIntegerField(name as string) as integer  
getDoubleField(name as string) as float  
getStringField(name as string) as string  
moveFirst()  
moveNext()  
moveLast()  
movePrevious()  
update()  
edit()  
delete()  
close()

**Contributors**

Jose L Cuevas	University of Puerto Rico at Mayaguez	jose@uprm.edu

This guide is work in progress please feel free to send changes, corrections and new content.