Intro

I recommend you to use MSDN (Microsoft Developer Network)!
This tutorial is about Visual Basic native code not P-CODE.
VISUAL BASIC is a high level programming language.
BASIC stands for "Beginners All-purpose Symbolic Instruction Code".

Types of data in Visual Basic:
1. Byte (UI1)
- can keep a value from 0 to 255 and is equivalent with ASCII code of one char, stored as 1 byte
2. Int (I2 - integer) (symbol: %)
- can keep a value from -32768 to 32767, stored as 2 bytes (word)
3. Long (I4 -long integer) (symbol: &)
- can keep a value from -2147483648 to 2147483647, stored as 4 bytes (dword)
4. Single (R4 - single real) (symbol: !):
Single-precision floating-point variables as 32-bit (4-byte) floating-point numbers, ranging in value
from -3.402823E38 to -1.401298E-45 for negative values, and 1.401298E-45 to 3.402823E38 for
positive values
5. Double (R8 - double real) (symbol: #):
Double-precision floating-point numbers as 64-bit numbers in the range -1.79769313486232E308 to 4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308
for positive values
6. String (Str/Bstr) (symbol: $)
- can store up to 65000 chars, used memory depend on string size
7. Bolean:
The bolean in Visual Basic has 2 value: True = 0FFFFh and False = 0.
Memory requirement: two bytes
8. Curency (Cy) (symbol: @):
It is a data type with a range of -922,337,203,685,477.5808 to 922,337,203,685,477.5807.
Use this data type for calculations involving money and for fixed-point

calculations where
accuracy is particularly important.
9. Date:
Eight bytes time information.
Range: 1/1/100 to 12/31/9999
A variant of type date is in Visual Basic is internally stored as a double (real number) - strange thing!
So when we have a date we have a qword, butt in functions with date will be only the offset of qword.
The variant of type date is composed by date+time, the date is the integer portion of the real number,
the fractional part is the time.
When in VB we have time the integer portion is zero.
Adding integers to a Date variable is equivalent to adding days, because the integer portion
of a Date variable represents the number of days that have passed since December 30, 1899.

10. Object:
A object can be: Instances of classes, OLE objects
Object variables are stored as 32-bit (4-byte) addresses that refer to objects within an application or within some other application.
Dim objDb As Object ' declare a variable as object
Set objDb = OpenDatabase("c:\Vb5\Biblio.mdb") ' assign the value to refer a object

When declaring object variables, try to use specific classes (such as TextBox instead of Control
or, in the preceding case, Database instead of Object) rather than the generic Object.
Visual Basic can resolve references to the properties and methods of objects with specific
types before you run an application. This allows the application to perform faster at run
time. Specific classes are listed in the Object Browser.

11. **Array**

Arrays allow you to refer to a series of variables by the same name and to use a number (an index)
to tell them apart. This helps you create smaller and simpler code in many situations, because you
can set up loops that deal efficiently with any number of cases by using the index number.
Arrays have both upper and lower bounds, and the elements of the array are contiguous within
those bounds. Because Visual Basic allocates space for each index number, avoid declaring
an array larger than necessary.
All the elements in an array have the same data type. Of course, when the data type is Variant,
the individual elements may contain different kinds of data (objects, strings, numbers, and so on).
In Visual Basic there are two types of arrays: a fixed-size array which always remains
the same size, and a dynamic array whose size can change at run-time.

12. Variant (any of the preceding data types):
Variant data type is a special data type that can contain any system-defined types of data:
numeric, string, or date, as well as user-defined types and the special values Empty and Null.
The Variant data type has a numeric storage size of 16 bytes and can contain data up to
the range of a Decimal, or a character storage size of 22 bytes (plus string length), and can
store any character text.
You don't have to convert between these types of data if you assign them to a Variant variable;
Visual Basic automatically performs any necessary conversion. For example:

```
Dim SomeValue ' variant by default or you can declare as Variant
SomeValue = "17" ' SomeValue contains "17" (a two-character string).
SomeValue = SomeValue - 15 ' SomeValue now contains the numeric value 2.
```

SomeValue = "U" & SomeValue ' SomeValue now contains "U2" (a two-character string)

The Empty Value:
A Variant variable has the Empty value before it is assigned a value.
The Empty value is a special value different from 0, a zero-length string (""), or the Null value!!!
If IsEmpty(Z) Then Z = 0 ' test if variant Z is Empty
The Empty value disappears as soon as any value (including 0, a zero-length string, or Null) is
assigned to a Variant variable. You can set a Variant variable back to Empty by assigning the keyword Empty to the Variant.

The Null Value:
The Variant data type can contain another special value: Null. Null is commonly used in database
applications to indicate unknown or missing data.
Z = Null ' set variant Z to Null
If you assign Null to a variable of any type other than Variant, a trappable error occurs.
Assigning Null to a Variant variable doesn't cause an error, and Null will propagate through
expressions involving Variant variables (though Null does not propagate through certain functions).
You can return Null from any Function procedure with a Variant return value.
Variables are not set to Null unless you explicitly assign Null to them, so if you don't
use Null in your application, you don't have to write code that tests for and handles it.

The Error Value:
In a Variant, Error is a special value used to indicate that an error condition has occurred in a
procedure. However, unlike for other kinds of errors, normal application-level error handling does
not occur. This allows you, or the application itself, to take some alternative

based on the error
value. Error values are created by converting real numbers to error values using the CVErr function.

When you want to see parameters of Visual Basic functions execute the code until EIP is equal
with address of the function, now look on the Stack Windows in Olly.
I sad this because Visual Basic has a lot of obfuscated code.

SmartCheck functions:
Val() - convert string to number
Str$() - convert number to string
**Left$**() - substring from left end e.g. Left$(Theodolite, 4) = "Theo"
Right$() - substring from right end
**Ltrim$**() - rtcLeftTrimBstr(string) - trim spaces off left e.g. Ltrim$ (" Hello ") = "Hello "
Rtrim$() - rtcRightTrimBstr(string) - trim spaces off right
Trim$() - rtcTrimBstr(string) - trim spaces off both ends
Mid() - rtcMidCharVar
Mid$(String:"araqmuP", long:1 - this is number of letter in the araqmuP string,
VARIANT:Integer:1 - this is number of chars the function must take)
In VB code this will look like this : letter = Mid$("araqmuP", 1, 1)
Asc() - rtcAnsiValueBstr - convert char to decimal ASCII value e.g. Asc("A") = 65
Chr$() - **rtcBstrFromAnsi** - convert ANSI code to char e.g. Chr$(65) = "A", returns a string
Hex - rtcHexVarFromVar
Len - __vbaLenBstr - length of string
Long - __vbaI2I4
OnError - __vbaOnError - used for error handling
LCase$(String) - rtcLowerCaseBstr - convert string to lowercase
UCase$(String) - rtcUpperCaseBstr - convert string to uppercase
Rnd() - rtcRandomNext - return a random number
ReDim - re-allocate memory or stuffs - that's not really important to us
IsNumeric returns Boolean:True - check if a string is a numeric value

Abs() - returns a value of the same type that is passed to it specifying the absolute value of a number
VariantChangeTypeEx - used to change the type of a variable, probably the variable will be comparated
LoadResData - load resources

Almost every string operation returns a Variant data type except Len.
For each of them, there is an identical function with a dollar sign ($) at the end of the function's name to indicate a String type return value. I recommend using the $ versions (such as Left$) whenever possible because they are more efficient.

Apis from OLEAUT32.DLL:
SysAllocStringByteLen(LPSTR:004103F0, DWORD:00000002) returns LPVOID:410434
Explanation:
String from **004103F0** is copied to new memory location 00410434
It is usually followed by __vbaStrVarVal(VARIATN:String"?") returns DWORD:410434

SysAllocStringLen(PTR:00000000, DWORD:00000029) returns LPVOID:410584
Allocate memory with size 29 and return a pointer to the memory

SysFreeString(BSTR:00410584)
Explanation:
- the string located at memory location 00410584 is cleared (free memory location 00410584)
These lines are especially good for "serial fishing" because when you click on them and look at the right window, you will see what Strings are being freed. Correct codes/serials are sometimes shown here.

VarNumFromParseNum:
- this procedure returns our dummy code converted into a double real number

Decompiling the VB code:

In VB the right address of variants (Var) are at address_showed-8. Strings (Str) have the right
address showed.
If we have as parameter a variant with subtype string this behaves as in variants (Var) case.

Example:
00402549 LEA ECX,DWORD PTR SS:[EBP-48]
0040254C LEA EDX,DWORD PTR SS:[EBP-58]
0040254F PUSH ECX ; ECX = address of number to be converted-8
; address of real variable = ECX+8
00402550 PUSH EDX ; address of destination string
00402551 MOV DWORD PTR SS:[EBP-40],0B ; set the value of number to be converted
00402558 MOV DWORD PTR SS:[EBP-48],2 ; the type of variable
0040255F CALL MSVBVM60.rtcHexVarFromVar

NuMega SmartCheck (great debugger) utilisation:
After you load the program, maximize the window, start the program (F5) **you must chose the "Show All Events" button**  **(to make SC to show all events) !!!**

Motto:
If Visual Basic **Then**
cracked at once

ThunRTMain:
Entry point of every VB program start with:
00401038 push offset EXEPROJECTINFO
0040103D call MSVBVM60:ThunRTMain

The ThunRTMain (the Thunder Runtime Engine) function exported by VB runtime DLL reads
the EXEPROJECTINFO structure and starts VB project initialization. The structure contains
a lot of important information, such as the address of the Main() function. The VB runtime
will eventually call this address (if it is defined).
More information in "VISUAL_BASIC_STRUCTURE".
For VB6:
734347D4 call dword ptr [esi+94] ; enter inside this one and you will find the main() function

API for comparing:

1. **__vbaStrCmp**(String:"xxxxx", String:"yyyyy") returns DWORD:0
- used for comparing strings "xxxxx" and "yyyyy"
- returns in ax 0 if comparated strings are equal else ax=0FFFFh (True)
- can be used to test if a string is empty by comparing string with Null string (00h)
__vbaStrTextCmp, **__vbaVarTextCmp**, **__vbaVarTextTst**, **__vbaStrComp**,
__vbaStrCompVar - use in the same way for comparing 2 strings
2.**__vbaVarTstEq**(VARIANT1, VARIANT2) returns DWORD:0
- is used to compare variants
- if they are the same ax=0FFFFh (True) else ax=0 (False), __vbaVarTstNe is the reverse function
- similar with __vbaVarCmpEq
3.
_vbaVarCmpEq/__vbaVarCmpGe/__vbaVarCmpGt/__vbaVarCmpLe/ /__vbaVarCmpLt/__vbaVarCmpNe

- is used to compare 2 variants
- Eq = equal; Ge = Greater or equal (>=); Gt = Greater (>); Le = Little or equal (<=); Lt = Little (<);
Ne = not equal
- if the condition is ok will set ax to 0FFFFh (True) else to 0 (False)
4. **__vbaFpCmpCy** - floating point compare currency
5. **__vbaInStrVar/__vbaInStr** - returns a long specifying the position of the first occurrence
of one string within another, if no occurrence returns 0
in VB: FindPos = InStr("KILLER", Chr$(13))
(translated to __vbaInStrVar)
in VB: FindPos = InStr("KILLER", "LL")
(translated to __vbaInStr)
InStr has two optional parameters: first parameter start position and last parameter comparation method.
Start position must start from 1 (1 is the first char).
rtcInStrRev - returns the position of an occurrence of one string within another, from the end of string
In VB: FindPos = InstrRev(string1,string2,start,compare)
6. **__vbaBoolVarNull** - tests if a variable is null, if is null returns 0 else returns 0FFFFh (-1),
- as parameter is pushed the right address of variable!
7. **rtcIsNull** - returns a Boolean value that indicates whether an expression contains no valid data (Null)
Null = no valid data
In VB:
```
Dim MyVar, MyCheck
MyCheck = IsNull(MyVar) ' returns False
MyVar = Null ' assign Null.
MyCheck = IsNull(MyVar) ' returns True
MyVar = Empty ' assign Empty
MyCheck = IsNull(MyVar) ' returns False
```
8. **rtcIsEmpty** - returns a Boolean value indicating whether a variable has been initialized, returns
TRUE if variant was initialized with Null
Empty = uninitialized
```
If IsEmpty(Z) Then Z = 0
```

9. **rtcIsNumeric** - returns a Boolean value indicating whether an expression can be evaluated as a number
In VB:
Dim MyVar, MyCheck
MyVar = 53 ' assign a value.
MyCheck = IsNumeric(MyVar) ' returns True.
MyVar = "459.95" ' assign a value
MyCheck = IsNumeric(MyVar) ' returns True
MyVar = "459.95 TEST" ' assign a value
MyCheck = IsNumeric(MyVar) ' returns False
rtcIsArray, **rtcIsDate, IsObject -** as rtcIsNumeric but test if is a array/date/object

API for strings:

1. **__vbaStrVarVal**(string, variant)
- convert variant to string and place them in a string
- ECX - source (variant), EDX - destination (string)
2. **__vbaStrCat**(string1, string2)
- concatenate string1 and string2
3. **__vbaLenVar**(receiver, variant)
- push in receiver length of variant
4.**__vbaLenBstr**(string) returns long
- returns in eax length of any string, in VB: len(string)
5. **__vbaLenBstrVar**(receiver, string)
- push in receiver length of string
6. **rtcUpperCaseBstr**(string)
- converts the string to upper case
In VB: UCase(string)
7. **rtcLowerCaseBstr**(string)
- converts the string to lower case
In VB: LCase(string)
8. int **MultiByteToWideChar**(
UINT CodePage, // code page
DWORD dwFlags, // character-type options
LPCSTR lpMultiByteStr, // address of string to map
int cchMultiByte, // number of characters in string

LPWSTR lpWideCharStr,   // address of wide-character buffer
int cchWideChar   // size of buffer
);
- MultiByteToWideChar converts a ASCII string to a UNICODE string
e.g. we have the string "Bob" witch is 426F6200h and after conversion will be "B.o.b.."
- 42006F00620000h. (Don't confuse the '.' with the full stop, I'm using it to represent the null character).
WideCharToMultiByte converts a UNICODE string to a ASCII string
__vbaStrToAnsi - convert a wide-char string (UNICODE) string to ASCII (ANSI)
__vbaStrToUnicode - convert a ASCII(ANSI) string to a wide-char string (UNICODE)
9. int **rtcMidCharVar**(string, length, position)
- e.g. we have string "1234567890", length = 2, position = 4 then the function returns a pointer
to string "45"
10. int **rtcLeftCharVar**(string, length)
- e.g. we have string "13567890", length = 3 then the function returns a pointer
to string "135"
11. int **rtcRightCharVar**(string, length)
- e.g. we have string "13567890", length = 4 then the function returns a pointer
to string "7890"
12. **rtcLeftCharBstr**(string, length) as String
- e.g. we have string "13567890", length = 4 then the function returns a pointer
to string "135"
13. **rtcAnsiValueBstr**(position, string) - convert to ANSI (number) the char witch specified position
from string and returns the number in eax
- for exemple Asc("A") = 65
14. **rtcHexVarFromVar**(buffer, number) - set buffer with a string representing the hexadecimal value
of a number
Example: 123 => 31 00 32 00 33 00 00 00

(wide chars, and just before the string, the length is stored, in this case: 2*3)
15. **rtcOctVarFromVar**(buffer, number) - set buffer with a string representing the octal value
16. **rtcLeftTrimBstr**(string) returns pointer to string - trim spaces off left
e.g. Ltrim$(" Hello ") = "Hello "
17. **rtcRightTrimBstr**(string) returns pointer to string - trim spaces off right
18. **rtcTrimBstr**(string) returns pointer to string - trim spaces off both ends
19. **rtcSplit -** split a string
In VB: Split(string_to_split,delimiter,limit as long,comparation_method)
Big example:

```
    Dim MyString, MyArray, Msg
MyString = "VBXisXfun!"
MyArray = Split(MyString, "x", -1, 1)
' now MyArray(0) contains "VB"; MyArray(1) contains "is" and
MyArray(2) contains "fun!".
MsgBox MyArray(0) ' -> will show "VB"
```

20. **rtcJoin** - join a string
In VB - lets join the string from 19. again:

```
MyString = Join(MyArray, "-") ' now MyString will contain "VB-is-fun!"
```

21. **rtcReplace** - replace from a string other string, returns a pointer to string
In VB:

```
Replace(expression,find,replacewith,start,count,comparation_method)
' last 3 parameters are optional
```

22. **rtcFilter** - returns a zero-based array containing subset of a string array based on a specified
filter criteria
In VB: Filter(sourcearray, match, include, comparation_method)
sourcearray - one-dimensional array of strings to be searched
match - string to search for
include - boolean value indicating whether to returns substrings that include or exclude match
    The array returned by the Filter function contains only enough elements to contain the number of
    matched items. No match -> empty array
23. **rtcStrReverse** - reverse a string

In VB: Str1 = StrReverse("KILLER") ' will result "RELLIK"
24. **__vbaLbound** - returns the index of the first element in the array
In VB: ilbound = LBound(array)
25. **__vbaUbound** - returns the index of the last element in the array
In VB: ulbound = UBound(array)
26. **__vbaLsetFixstr** - left aligns a string within a string variable, or copies a variable of one
user-defined type to another variable of a different user-defined type, LSet replaces any leftover characters in stringvar with spaces
In VB:
Dim MyString
MyString = "0123456789" ' Initialize string.
Lset MyString = "<-Left" ' MyString contains "<-Left ".
27. **__vbaFreeStr**
- frees a temporary string - set the pointer of string to 0
- __vbafreestr accept only one argument which is the address of the string to be deleted - this argument is ALWAYS passed through ECX
28. **__vbaFreeStrList**(number_of_strings, String1, String2,...) - free a list of temporary strings
29. **rtcSpaceVar** - returns a string consisting of the specified number of spaces
In VB:
Dim MyString
MyString = Space(10) ' returns a string with 10 spaces
30. **rtcStringVar** - returns a repeating character string of the length specified
In VB:
Dim MyString
MyString = String(5, "*") ' returns "*****"
MyString = String(5, 42) ' returns "*****"
MyString = String(10, "ABC") ' returns "AAAAAAAAAA"
31. **rtcStrConvVar2** - returns a Variant (String) converted as specified (can convert a string to lower-case)
Syntax: StrConv(string, conversion, LCID)

API for conversion:
Val() - convert string to number, **Str$**() - convert number to string,

CBool(expression) - convert expression to boolean;
CByte,CCur,CDate,CDbl, CInt, CLng, CSng, CStr, Cvar,CVErr(convert to error) - self explain
Asc() - **rtcAnsiValueBstr** - convert first char from a string to decimal ASCII value e.g. Asc("A") = 65
Chr$() - **rtcBstrFromAnsi** - convert ANSI code to char e.g. Chr$(65) = "A", returns a string
__vbaR8Str/rtcR8ValFromBstr - convert string to double (real)
__vbaStrR8 - convert double (real) to string
__vbaStrI2 - convert integer to string
__vbaStrI4 - convert long to string
__vbaI2Str - convert string to byte or integer
__vbaI4Str - convert string to long
__vbaR4Str - convert string to single (real)
__vbaR8Str - convert string to double (real)
__vbaI2I4 - convert long to integer (same number)
__vbaUI1I2 - convert integer to byte
vbaI4Var - convert variable to long
VarBstrFromI2 - convert integer to string
VarCyFromStr - convert string to currency
rtcBstrFromFormatVar - convert a variable to a string with a specified format (in SC: format$)
Format has a lot of useful stuff:

```
Dim MyTime, MyDate, MyStr
MyTime = #17:04:23#
MyDate = #January 27, 1993#
' Returns current system time in the system-defined long time format:
MyStr = Format(Time, "Long Time")
' Returns current system date in the system-defined long date format:
MyStr = Format(Date, "Long Date")
MyStr = Format(MyTime, "h:m:s") ' Returns "17:4:23".
MyStr = Format(MyTime, "hh:mm:ss AMPM") ' Returns "05:04:23 PM".
MyStr = Format(MyDate, "dddd, mmm d yyyy") ' Returns "Wednesday,'
Jan 27 1993".
' If format is not supplied, a string is returned.
MyStr = Format(23) ' Returns "23".
```

' User-defined formats:
MyStr = Format(5459.4, "##,##0.00") ' Returns "5,459.40".
MyStr = Format(334.9, "###0.00") ' Returns "334.90".
MyStr = Format(5, "0.00%") ' Returns "500.00%".
MyStr = Format("HELLO", "<") ' Returns "hello".
MyStr = Format("This is it", ">") ' Returns "THIS IS IT".

API for data:
1. Functions which copy data from a place to other:
You should notice that strings are not really copied to a new location only the reference of them.
So basically these functions copy the reference of a string to a new location. After copy the old location is released with **__vbaFreeVar/__vbaFreeStr**

__vbaVarMove(VARIANT:integer:13, VARIANT:Integer:6) - this move the final result (13) where
the value 6 was written (overwriting the value 6):

LEA EDX,DWORD PTR SS:[EBP-38]
PUSH EDX ; at edx+8 is the offset of string
CALL MSVBVM60.**__vbaStrVarMove ;** returns offset of string
MOV EDX,EAX ; source
LEA ECX,DWORD PTR SS:[EBP-18] ; destination
; for almost all function edx = destination, ecx = source (not like in this example)
CALL MSVBVM60.**__vbaStrMove** ; copy source into destination

__vbaStrCopy - copy string to memory
__vbaVarCopy or **__vbaVarMove** - copy variant to memory
__vbaVarDup - copy variant from source to destination (a new location for string)
EDX = source, ECX = destination
Also exist function for copying data for other variable type: __vbaAryMove

2. **__vbaVarCat** - join 2 variants together
Exemple: __vbaVarCat(VARIANT:String:"aa", VARIANT:String:"bb") returns DWORD:63F974

- join "bb" to "aa" to form "aabb"
- this can even add to a string a variant
3. **__vbaVarForInit(VARIANT:Empty, PTR:0063F920, PTR:0063F91.....)**
- prepare for() cycle (set a For...Next Loop)
There will usually be **__vbaVarForNext** somewhere below as well
4. **__vbaVarForNext** - used in codes where there is a For... Next... Statement (Loop)
5. Releasing local variables (set them to 0):
__vbaFreeVar - frees a temporary variable, accepts only 1 Argument, which is the address of the
variable to be deleted, this argument is ALWAYS passed through ECX
- uses the API function __imp_SysFreeString() from OLEAUT32.DLL that carries out the actual deallocation of a variable
__vbaFreeVarList(number_of_var, offset_var1, offset_var2, ...) - free a list of temporary variables
6. **VarPtr**(variant) returns a long - get the pointer of variant
7. **__vbaRedim** - used at procedure level to reallocate storage space for dynamic
array variables
In VB:
Dim alngNum() As Long
ReDim alngNum(0 To 99)
' redimensionate the alngNum to a array that contains 100 elements
If you use after ReDim the keyword Preserve you will keep the old contain of array
and the function used will be **__vbaRedimPreserve**
8. **rtcArray** - returns a Variant containing an array
In VB:
Dim A
A = Array(10,20,30)
9. In VB: **Erase** array - reinitializes the elements of fixed-size arrays and deallocates dynamic-array
storage space, will be translated into **__vbaErase** followed by **__vbaAryDestruct**
In VB:
Dim NumArray(9) ' construct a array - with **__vbaAryConstruct2**

Dim DynamicArray()
ReDim DynamicArray(9) ' allocate storage space
Erase NumArray ' each element is reinitialized
Erase DynamicArray ' free memory used by array
10. **__vbaGenerateBoundsError** - is called when the size of array is overflow and generates an error
Arrays have both upper and lower bounds, and the elements of the array are contiguous within those bounds
Visual Basic allocates space for each index number, avoid declaring an array larger than necessary
11. **Special formatting:**
rtcFormatCurrency - FormatCurrency; rtcFormatDateTime - FormatDateTime; rtcFormatNumber - FormatNumber; rtcFormatPercent - FormatPercent
12. **rtcTypeName** - returns a string that provides Variant subtype information about a variable
In VB:
MyType = TypeName("VBScript") ' returns "String"
13. **rtcVarType** - similar with rtcTypeName butt returns a value indicating the subtype of a variable

Math:
__vbaVarAdd(TARGET, VARIANT1, VARIANT2) - add VARIANT1 to VARIANT2 and put him
in TARGET
__vbaVarSub(VARIANT1, VARIANT2) - sub from VARIANT1 VARIANT2
__vbaVarMul(VARIANT1, VARIANT2) - multiply VARIANT1 with VARIANT2
__vbaVarDiv(VARIANT1, VARIANT2) - divide VARIANT1 with VARIANT2
__vbaVarIdiv - divide (like before) butt the result will be a integer
__vbaVarXor(VARIANT1, VARIANT2) - XOR VARIANT1 with VARIANT2
__vbaVarOr(VARIANT1, VARIANT2) - OR VARIANT1 with VARIANT2
rtcSqr - square from a number, in VB is without rtc

rtcAbs - returns a value of the same type that is passed to it specifying the absolute value of a number
rtcCos - returns a double specifying the cosine of an angle
rtcAtn - returns the arctangent of a number
rtcLog - returns the natural logarithm of a number
rtcRandomize - it's used to initialize the random numbers engine, in VB: Randomize
rtcRandomNext - returns a random real number between 0 and 1, in VB: Rnd
rtcExp(power) - returns e (the base of natural logarithms) raised to a power
rtcRound(VARIANT) - rounds a variant e.g.: Round(5.5) => 6
__vbaPowerR8(number1,number2) - pow 2 real numbers: number1 ^ number2
FnLenVar - returns the length of variable
__vbaR8Sgn - returns an integer indicating the sign of a number;
Sgn(number) returns 1 if number >0; 0 if number = 0 or -1 if number <0
__vbaFPInt, **__vbaFPFix** - returns the integer portion of a number;
- the difference between Int and Fix is that if number is negative, Int returns the first negative
integer less than or equal to number, whereas Fix returns the first negative integer greater than
or equal to number, In VB: Int/Fix

API for file:
rtcFreeFile = free file returns a filenumber that isn't in used, it is an integer and can be used
to open files (in VB: hfile = FreeFile)
__vbaFileOpen - open a file
Translated in VB:
Open "c:\projectana\superpa.lic" for input as hfile
__vbaLineInputStr - read a line from file in a string; in VB: Line Input #hfile, BufferAddress
__vbaInputFile - reads data from files and stores the file data in your program's variables and controls
In VB: Input #hfile, BufferAddress1,BufferAddress2...
__vbaPrintFile - write in a file, similar syntax with Input

__vbaGet3 - reads data from an open disk file into a variable
In VB: Get hfile,long_reading_begins,variable
__vbaPut3 - writes data from a variable to a disk file
In VB: Put hfile,long_writing_begins,variable
rtcFileLen, **rtcFileLength** - returns the length of file (in VB: FileLength = LOF(hfile))
rtcEndOfFile - will returns in ax 0 if is not end of file else 0FFFFh (-1)
__vbaFileClose - close a file (in VB: Close hfile)
__vbaFileCloseAll - close all file (in VB: Close)
rtcFileSeek(hfile,position) - set and get the current position within a file
rtcKillFiles - delete a file (in VB: Kill "name_of_file")
rtcFileReset - the Reset statement closes any open files and writes the contents to disk
- this statement operates the same as the Close statement without parameters
rtcFileLocation - this function returns the current position of the file pointer within a file opened
with the Open statement (in VB: Position = Loc(hfile))
rtcFileAttributes - get or set the file attribute (in VB: iMode = FileAttr(hfile, 1)
- this will get the mode for which the file is opened)
__vbaFileLock - controls access by other processes to all or part of a file opened using
the Open statement
In VB: Lock/Unlock hfile,recordrange ' recordrange = the range of records to lock or unlock
rtcFileCopy - copy a file
In VB: FileCopy "source", "destination"
__vbaNameFile - rename a file
In VB: Name "old" As "new"

API for registry:
rtcDeleteSetting - delete a registry setting as well as the registry key
In VB: DeleteSetting capplication, csection,Key
capplication - this string setting, which is contained in the upper key of the applications setting,
is one that you would like to delete

csection - this string setting contains the section of the registry that contains the key that you want to delete
Key - this optional parameter contains the name of the key that you want to delete; if this parameter is left empty, all of the keys in the named section are deleted
If the named section or key does not exist, the DeleteSetting function takes no action, nor does it
returns an error
Ex.: DeleteSetting "API Viewer", "Options","View"
rtcGetSetting - allows you to retrieve the value of a specified registry key
In VB: GetSetting capplication, csection, key, cDefault
capplication - this string setting contains the upper key of the applications setting that you would
like to retrieve
csection - this string setting contains the section of the Registry that contains the key you want
to retrieve
key - this string setting contains the name of the key whose value you want to retrieve
cDefault - this parameter is used to supply your application with a default value if the key in the
specified section does not exist (optional)
The GetSetting function returns a Variant value containing the value of the key; if the key is not found,
the cDefault value is returned
Ex.: vValue= GetSetting("API Viewer", "Options", "View","")
rtcGetAllSettings - allows you to retrieve all of the Registry keys and their settings for a specified
application
In VB: GetAllSettings capplication, csection
capplication - this string value contains the upper key of the applications setting that you would
like to retrieve
csection - this string setting contains the name of the section of the registry whose values you
want to retrieve
The GetAllSettings function returns a two-dimensional array of Variant

values. This array contains the
keys as well as the values from the named section of the Registry. If the value is not found, the function
returns a Null Variant value.
Ex.: vValue= GetAllSettings("API Viewer", "Options")
rtcSaveSetting - allows you to save a key and its value to the Registry (If the key did not previously
exist, it is created.)
In VB: SaveSetting capplication, csection, key, value
value - a string witch specify new value

Date:
In VB a date is stored as double (qword)
rtcGetTimer - returns a single representing the number of seconds elapsed since 12:00 AM (midnight)
In VB: Timer
rtcGetDateVar/rtcGetPresentDate - read current system date
In VB: Date
rtcGetDateBstr - returns offset of a string with current date
rtcGetTimeVar - reads current system time
In VB: Time
rtcGetTimeValue - returns a variant (Date) containing the time
In VB: TimeValue(time)
rtcGetCurrentCalendar - get the current calendar - used in rtcGetPresentDate; any instant in time
can be represented as a set of numeric values using a particular calendar
rtcDateVar - copy a date value from a qword (double) to st(0),
the offset must be fixed with +8 to see the real date value to be set
rtcSetDateVar - set a variable of type date with a value
rtcFileDateTime - returns a Variant (Date) that indicates the date and time when a file was
created or last modified, in VB: FileDateTime(pathname)
__vbaDateStr - has as parameter a unicode string with the date, set st(0) to date from the string
rtcDateAdd - returns a date to which a specified time interval has been added
In VB: NewDate = DateAdd("m", 1, Date) ' add 1 to month to Date

rtcDateDiff - returns the number of intervals between two dates
DiffADate = "Days from today: " & DateDiff("d", Now, theDate) ' returns the number of days ("d") between
current date and other date
rtcDatePart - returns the specified part of a given date
GetQuarter = DatePart("q", TheDate) ' displays the quarter of the year in which it occurs
rtcPackDate - returns a Variant of subtype Date for a specified year, month, and day
In VB:
Dim MyDate1, MyDate2
MyDate1 = DateSerial(1970, 1, 1) ' returns January 1, 1970
MyDate2 = DateSerial(1990 - 10, 8 - 2, 1 - 1) ' returns May 31, 1980
rtcPackTime - in VB: TimeSerial(time) - same thing with rtcPackDate butt for time
rtcGetDateValue - returns a Variant of subtype Date
In VB:
Dim MyDate
MyDate = DateValue("September 11, 1963") ' returns a date
rtcGetDayOfMonth - returns a whole number between 1 and 31, inclusive, representing the day of the month
In VB:
Dim MyDay
MyDay = Day("October 19, 1962") ' MyDay contains 19
rtcGetHourOfDay - Hour(time), **rtcGetMinuteOfHour** - Minute(time), **rtcGetMonthOfYear**
- Month(time), **rtcGetSecondOfMinute** - Second(time),
rtcGetDayOfWeek - GetDayOfWeek(time),
rtcGetYear - Year(time) - self explain
rtcMonthName,rtcWeekdayName - returns a string indicating the specified month/weekday
In VB:
Dim MyVar
MyVar = MonthName(10, True) ' MyVar contains "Oct"

Other API:
rtcChoose - selects and returns a value from a list of arguments

In VB: Choose(index, choice-1[, choice-2, ... [, choice-n]])
Choose returns a value from the list of choices based on the value of index
rtcCurrentDir - returns a Variant (String) representing the current path
In VB: CurDir[(drive)]
rtcDir - returns a string representing the name of a file, directory, or folder that matches a
specified pattern or file attribute, or the volume label of a drive
In VB: Dir(pathname, attributes)
Exemples:
MyFile = Dir("C:\WINDOWS\WIN.INI") ' returns "WIN.INI" if it exists
MyFile = Dir("C:\WINDOWS*.INI") ' returns the name of first file with ini extension
rtcAppActivate - makes a windows focus
In VB: AppActivate "title", wait_time
rtcSendKeys - sends key strokes to current application
In VB: SendKeys "string",wait
rtcChangeDir - change the current directory
In VB: ChDir "path"
rtcMakeDir - makes a directory
In VB: MkDir "path"
rtcRemoveDir - removes a directory
In VB: MkDir "path"
RtcMsgBox(VARIANT:String:"Nope! That's not right", Integer:0,VARIANT:String:"Wrong",VARIANT.....)
- create a MessageBox with title "Wrong" and message "Nope! That's not right"
rtcBeep - beep on internal speaker, in VB: Beep
__vbaNew2(ADDR offset FormDescriptor_0_, ADDR Form1Instance) - create a new VB form,
call the routine Form.Initialize
rtcInputBox - displays a prompt in a dialog box, waits for the user to input text or click a button,
and returns a String containing the contents of the text box
In VB:
Serial$ = InputBox("Enter the working code given by Trends", "Enter Working Code")
rtcShell - runs an executable program and returns a Variant (Double)

representing the program's task
ID if successful, otherwise it returns zero
In VB: Shell(pathname[,windowstyle])
rtcCommandVar - reads command line (like GetCommandLineA)
In VB: String CommandLine = Command
__vbaHresultCheckObj - checks the result of any operation made with a object; the operation can be
closing of the About box, after the timer has elapsed, this usually come after a operation (command)
made with a object
__vbaExceptHandler - exception handler for Visual Basic, this one is called when we have a bug,
In VB: On Error GoTo CantReadF1
__vbaSetSystemError - has a parameter witch holds system code
__vbaExitProc - called at the end of a procedure
__vbaEnd - close the program
rtcQBColor(color) - set the color, translated from QBColor(color)
rtcRgb - returns a long whole number representing an RGB color value
Syntax: RGB(red, green, blue)
rtcVarFromError - returns the error message that corresponds to a given error number
Syntax: Error[(errornumber)]
__vbaSetSystemError(system_code) - set the system error code
rtcSwitch - evaluates a list of expressions and returns a Variant value or an expression
associated with the first expression in the list that is True.
Syntax: Switch(expr-1, value-1[, expr-2, value-2 … [, expr-n,value-n]])
rtcDoEvents - yields execution so that the operating system can process other events
rtcEnvironVar - returns the String associated with an operating system environment variable

API for objects:
1. **__vbaVarSetVar** - assigns an object reference to a variable or property, or associates a procedure reference with an event
In VB:
Dim fso ' define a variant

Set fso = CreateObject("Scripting.FileSystemObject")
' CreateObject - creates and returns a reference to an ActiveX object
' Syntax is: CreateObject(class,[servername])
Will become:
rtcCreateObject2(local_var,class,servername)
__vbaVarSetVar(receiver,local_var) ' set receiver as local_var
2. **__vbaVarSetObjAddref/__vbaObjSetAddref**
In VB:
Dim MyObject
Set MyObject = Nothing ' the Nothing keyword is used to disassociate an object variable
' from any actual object
3. **rtcCallByName** - executes a method of an object, or sets or returns a property of an object,
Syntax: CallByName(object, procname, calltype,[args()])
In VB:
CallByName Text1, "MousePointer", vbLet, vbCrosshair
Result = CallByName (Text1, "MousePointer", vbGet)
CallByName Text1, "Move", vbMethod, 100, 100
' where Text1 is the name of one TextBox (object)
4. **rtcGetObject** - returns a reference to an object provided by an ActiveX component
Syntax: GetObject([pathname] [, class])
Use the GetObject function to access an ActiveX object from a file and assign the object to an
object variable
Dim MyObject As Object
Set MyObject = GetObject("C:\DRAWINGS\SAMPLE.DRW", "FIGMENT.DRAWING")
' In the example, FIGMENT is the name of a drawing application and DRAWING is one of the
' object types it supports

_TextBox:: AddRef returns DWORD:1 - **AddRef** increment reference count of a object (instantiation)
Zombie_AddRef Function takes the Object Reference.
In this function the parent object (in this case Form) is passed as a

parameter
and uses AddRef to increment reference count of the object (instantiation). Since COM objects are responsible for their lifetime, the resources they use are
allocated until the reference count is 0, when it reaches 0 the objects enter zombie state & can be deallocated to free resources.

5. **__vbaObjSet** - set a object variable
00401C52 MOV EAX,DWORD PTR DS:[ESI] ; use current instance
00401C54 PUSH ESI ; Form1 instance
00401C55 CALL DWORD PTR DS:[EAX+304] ; get the ID of Text2, if is Text1 would be 2FC
00401C5B LEA ECX,DWORD PTR SS:[EBP-20]
00401C5E PUSH EAX ; the object identifier
00401C5F PUSH ECX ; address of a variable witch will keep the object identifier
00401C60 CALL MSVBVM60.__vbaObjSet ; set a object variable, will returns in eax the new value of object
00401C66 MOV ESI,EAX
00401C68 LEA EAX,DWORD PTR SS:[EBP-1C]
00401C6B PUSH EAX
00401C6C PUSH ESI ; the object identifier
00401C6D MOV EDX,DWORD PTR DS:[ESI]
00401C6F CALL DWORD PTR DS:[EDX+A0] ;
; This call is a Visual Basic command
; we have 0A0 so the command is Text2.Text - get text from Text1
; Can be used other register
rtcErrObj - no parameter, returns offset of a error object, after that will be set a object with the
returned value
6. **__vbaFreeObj** - free an object variable (set object to 0)
- push in ecx the address of object to be free
7. **__vbaFreeObjList**(number_of_objects, Object1,Object2,...) - free a list of objects (variables)

VB Commands

How VB commands looks:
push 00 ; the value to be set or offset of variable witch keep the result
push eax ; object instance
mov edx, dword ptr [eax]
call dword ptr [edx+000001BC] ; set a From visibility to False
(NagForm.Visible = False)

Visual Basic 6.0 commands reference:
00010h - unload a form, in VB "Unload Form1"
00050h - read the aplication path, in VB App.Path
Exemple: hfile = FreeFile
Open App.Path + "\data.cfg" For Input As hfile
00054h - changing the caption property for any object which has one. An example of how this knowledge
may be useful is if a program has an annoying "Not Registered" in the title bar of a form;
- this is used even when you change the caption of any label
(mainForm.Caption = "Blah Blah v1.0 - Unregistered")
00064h - set color of background form
00074h - enabling/disabling a menu item (mnuRegister.Enabled = True)
0008Ch - set state of a button (Command1.Enabled = False or True)
00094h - making a command button visible/invisible (Command1.Visible = True)
0009Ch - making labels visible/invisible (Label1.Visible = True)
000A0h - get text for a TextBox
000A4h - set text for a TextBox
000E0h - read state of a Radio buton (Option1.Value)
01BCh - set the form visibility (NagForm.Visible = True or False)
02B0h - show a form (Form.Show)
02B4h - hide a form (Form.Hide)
0204h - Text1.SetFocus - set focus Text1
0040h - SavePicture Image, "TEST.BMP" ' save picture to file
0044h - LoadPicture("PARTY.BMP")
0104h - Set Command1.DragIcon = LoadPicture("MYICON.ICO") ' set

drag icon
0164h - Set Form1.Icon = LoadPicture("MYICON.ICO")
0058h - Clipboard.SetData LoadPicture("PARTY.BMP")
0060h - Clipboard.SetText "Text"
000Ch - File1.FileName = "*.*"

PICTURE BOX:
0154h - **Set Picture** = LoadPicture("PARTY.BMP")
0288h - Form1.Picture1.PSet (X,Y) - draw a point
028Ch - Form1.Picture1.Scale (X1, Y1)-(X2, Y2) - scale the image
027Ch - Form1.Picture1.Line(X1, Y1)-(X2, Y2)- draw a line