

Russian → [English](#) ▾

VBStreets conference

The whole taste of programming!

[Skip](#)

 [Advanced search](#)

- [Board index](#) < [Knowledge base](#) < [VB Internals](#)
- [Change font size](#)
- [FAQ](#)
- [Registration](#)
- [Entrance](#)

[§ 10. Ruby + EB = Visual Basic](#)

Moderator: [Hacker](#)

[Reply](#)

Posts: 22 • Page 1 of 1

[Hacker](#)

Telepath



Messages: 16416

Registered: 11/13/2005 (Sun) 2:43

From: Kazakhstan, Petropavlovsk

- [Website](#)
- [ICQ](#)

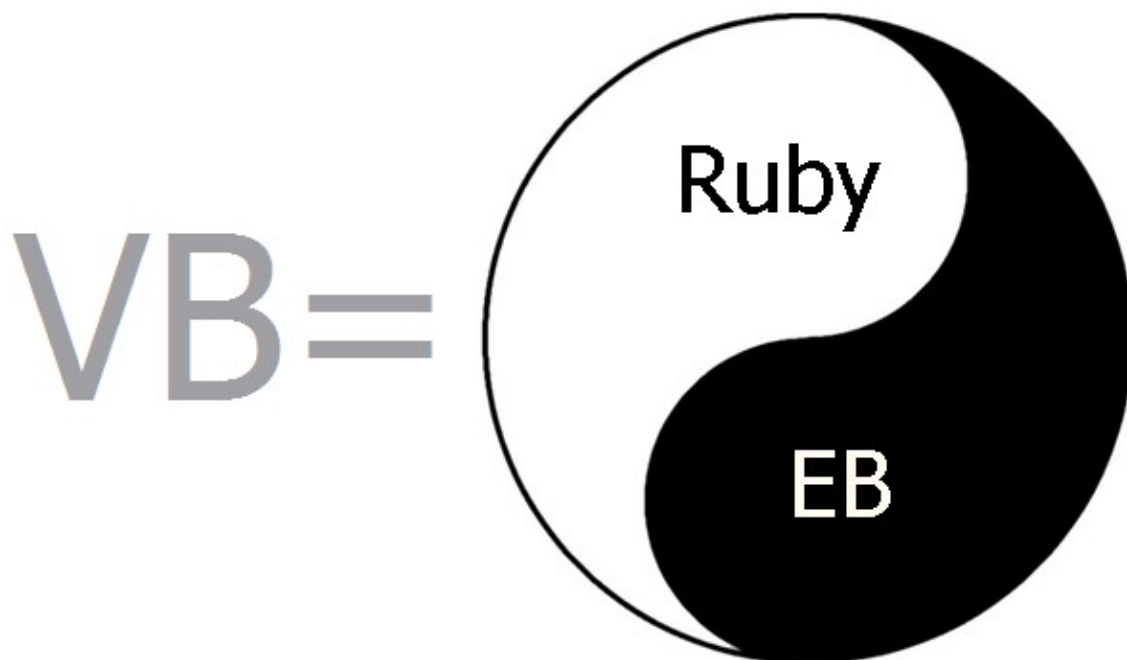
[§ 10. Ruby + EB = Visual Basic](#)

▫ [Hacker](#) » 06/10/2019 (Mon) 14:01

Before my serious immersion in the study of the inner world and the VB6 device, I, like many, probably, had the following idea in my head: *VB is the main product, while VBA is a by-product, the result of cutting the main product in terms of functionality and attaching to Office applications such as Excel, Word.* But it often happens that the layman's idea of the structure of things, built on the basis of attempts to predict this structure, can radically

differ from the true structure. And in this case, this is exactly our case: in fact, everything turned out to be quite the opposite, and not VBA is a by-product of the creation of VB, but rather VB is a by-product of the appearance of VBA.

This article will shed some light on what two large parts Visual Basic consists of, what is the ideological background and what kind of people are behind the appearance of each of them into the world.



As you might have guessed from the above, there are two big components. These are **Ruby** and **EB**. Perhaps the most apt statement about all this: *Visual Basic is the result of the "marriage" of **Ruby** and **EB** - two technologies that appeared independently of each other, and very well suited to each other.*

Ruby

A very important note to make before we go any further: here and below the word "Ruby" has nothing to do with a programming language [Ruby](#) and the [Ruby on Rails framework](#). After all, the word "ruby" is simply translated as "ruby".



(Pictured: Alan Couper)

If you've ever searched for information about how Visual Basic came into being, you've probably stumbled across Wikipedia or somewhere else with words like this:

May 1991 - Visual Basic 1.0 for Microsoft Windows released. The QBasic syntax was taken as the basis of the language, and the innovation, which then brought great popularity to the language, was the principle of communication between the language and the graphical interface. This principle was developed by [Alan Cooper](#) ([Alan Cooper](#)) and implemented in the prototype Tripod (also known as [Ruby](#)). The first Visual Basic was an interpreter.

From an article about [Alan Cooper](#) himself , you can learn that he, in particular, is known as the "father of Visual Basic." [The English version](#) of the article is a bit more informative.

In fact, Alan Cooper did not work at Microsoft, was not on the VB development team - his merit is elsewhere. Alan Cooper came up with the concept of easy creation of visual skins and, together with colleagues, created a tool for building visual skins. Let me remind you that we are talking about the 80s and the time when OS with graphical interfaces was just gaining popularity, and the mainstream was DOS and other OS with a text interface.

The product that Alan Cooper's team made was called Ruby, and the most interesting thing is that **it was not a tool for programmers** - it was a tool for ordinary users without special skills, giving them, according to Cooper's idea, the ability to quickly dazzle a visual shell that would facilitate some kind of work. A kind of **designer of visual interfaces** for a wide range of people. Oh, this is the time of romantics, who believed that it is worth giving people a computer - and everyone will learn and begin to write programs with their own hands to make the computer do exactly what the owner needs. Ruby was neither a programming language nor an integrated development environment (IDE) in the modern sense.

According to Cooper, when he saw Windows 1.0, he realized that this platform had a great future. He was pleasantly struck by two things: the graphical interface and the concept of a DLL, which allows you to create dynamically configurable extensible systems, but at the same time, the then Windows shell (Explorer and the desktop in the modern sense did not yet exist) seemed to him simply terrible. And then he decided to write his ideal shell for Windows. Talking to his clients and trying to figure out what the ideal shell should be, Alan Cooper realized that the perfect shell does not exist. And that it is necessary to give users a tool that will allow each of them to design their own shell, tailored to their needs and their tasks, and not try to push them some ideal shell, while trying to explain to them what its ideal is.

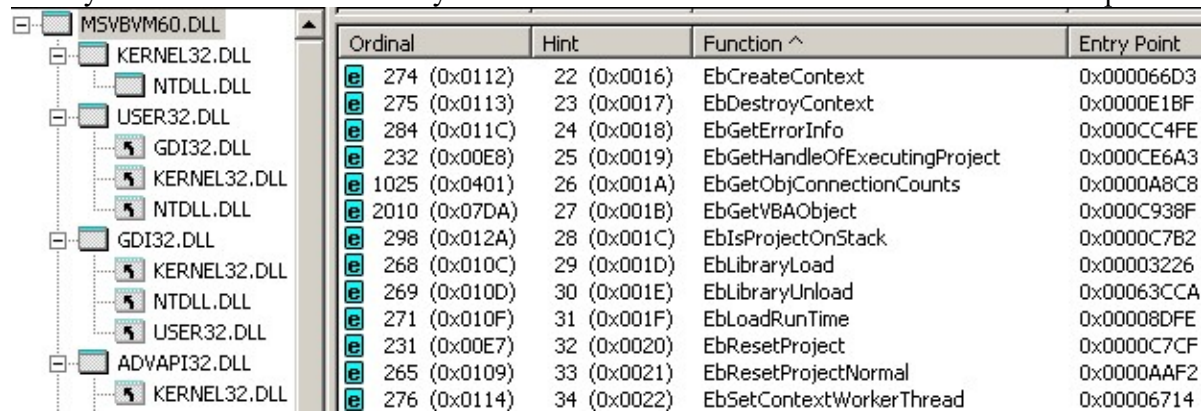
Later, Alan Cooper showed his development to Bill Gates. Gates liked the concept that the user can draw visual controls, place them anywhere on the workspace, drag and drop, resize and, most importantly, **associate some actions with controls**, he liked so much that he decided to buy Ruby. From now on, Microsoft became the owner of Ruby and determined the future fate of the product.

Microsoft decided to change the essence of Ruby: leaving the concept of easy construction of visual interfaces (“draw a button where necessary and quickly bind some action to it”), from a tool for creating visual shells (over existing programs) with a target audience consisting of ordinary users, turn it into a tool for creating new programs with visual interfaces, the target audience of which would be only programmers. Let me remind you that initially Ruby, in addition to the principle of visual design, did not have any programming language (there was only a simple system of commands, the set of which was extended by plug-in DLLs, as well as a set of controls in the palette (Cooper called them "gizmo", which in translated into Russian - “thingy”, this term was also preserved in VB6), which also expanded due to third-party DLLs), and it was necessary to decide which language would form the basis of the new instrument. But Microsoft already had QBASIC, and you can guess exactly what plans were born in Bill Gates's head when he decided to buy such a promising asset as Ruby.

At this point in history, the concept and mechanism for building visual interfaces, completely devoid of any programming language behind it, had to meet with a programming language and a mechanism that provides its execution, completely devoid of a visual component.

EB

Have you ever looked at how many different undocumented functions the VB runtime exports?



Ordinal	Hint	Function ^	Entry Point
274 (0x0112)	22 (0x0016)	EbCreateContext	0x000066D3
275 (0x0113)	23 (0x0017)	EbDestroyContext	0x0000E1BF
284 (0x011C)	24 (0x0018)	EbGetErrorInfo	0x000CC4FE
232 (0x00E8)	25 (0x0019)	EbGetHandleOfExecutingProject	0x000CE6A3
1025 (0x0401)	26 (0x001A)	EbGetObjConnectionCounts	0x0000A8C8
2010 (0x07DA)	27 (0x001B)	EbGetVBAObject	0x000C938F
298 (0x012A)	28 (0x001C)	EbIsProjectOnStack	0x0000C7B2
268 (0x010C)	29 (0x001D)	EbLibraryLoad	0x00003226
269 (0x010D)	30 (0x001E)	EbLibraryUnload	0x00063CCA
271 (0x010F)	31 (0x001F)	EbLoadRunTime	0x00008DFE
231 (0x00E7)	32 (0x0020)	EbResetProject	0x0000C7CF
265 (0x0109)	33 (0x0021)	EbResetProjectNormal	0x0000AAF2
276 (0x0114)	34 (0x0022)	EbSetContextWorkerThread	0x00006714

Surely you must have heard about the [EbExecuteLine](#) function, which allows you to interpret and execute an arbitrary line of code (though only when working under an IDE). Have you ever wondered what the "Eb" prefix in the names of all these functions means? How is it decrypted?

Now we will deal with this.

I am not sure exactly which of the two versions of the two possible decryptions is more correct, but, obviously, both versions are not far from the truth. Let's start with the first version and turn to one of the articles of the notorious [Joel Spolsky](#) for this.



The article is called "[My First BillG Review](#)" ("My first check by Bill G.", there is a [Russian translation](#)).

Brief retelling of history:

In the summer of 1991, Joel got a job at Microsoft in the Excel development department. One of the actual problems of Excel at that time was that Excel had its own poor programming language for macros, in which there were no global and local variables, there were no procedures, there was a goto, but there were no labels. The language did not have its own name, and was conditionally called "Excel macros".

Joel's task was to solve this problem. There was an opinion that the solution to the problem should be somehow connected with the Basic language. In another department (within Microsoft) there was already an operating time to create an object-oriented BASIC, code-named "Silver". The Silver team manager saw only one use for his technology—Excel. Joel convinced the people on the Basics team that the Excel team needed something like *Basic for Excel* . At the same time, he managed to insist that 4 features should be added to the language:

1. Add Variant variables that can store values of any other types, otherwise it would be impossible to store the value of an Excel table cell in a variable (at least without resorting to Select Case).
2. Add late binding (via IDispatch) because the original Silver architecture required a deep understanding of the type system, something that Excel macro writers wouldn't want to understand.
3. Add a For Each construct borrowed from [csh](#) .
4. Add a With ... End With construct , borrowed from Pascal.

After that, Joel sat down and wrote a specification for the future language **Excel Basic** , which took about **500 pages** , which was then sent to Bill Gates for review / verification. The remainder of the article describes the subsequent meeting with B. Gates, in which he asked Joel questions based on marginal notes Gates left when reading the specification.

The final part of the article says that over time the state of affairs has advanced a lot, **Excel Basic became known as Visual Basic for Applications**, and Gates once remembered how difficult it was to hire that smart manager in the Excel team who did all this.

So, as you can understand from the text of the article, at some point the need to replace the poor macro language in Excel with something more advanced led to the fact that the bet was made on basic-like languages, and based on the developments available in another department for creating object-oriented basic-like language, **Excel Basic** (abbreviated as **EB**) was created - an object-oriented general-purpose language suitable for automating almost

anything, and therefore quickly went beyond pure Excel in terms of applicability. Since the language allowed writing macros / extensions / automation not only for Excel, but for anything, the technology began to be used not only in Word / Access - Microsoft managed to sell it to the creators of [AutoCAD](#) , [SolidWorks](#) , [Corel Draw](#) , [ArcGIS](#) . It is understandable that a technology with such a broad scope could no longer be called **Excel Basic** and was renamed **Visual Basic for Applications** , especially when you consider that in addition to programs included in Office and third-party programs, this technology has become an integral part of what is known as simply **Visual Basic** . At the same time, the abbreviation "EB" took root in the name of many functions and structures related to technology, where it remains to this day.

The second version of the decoding of the name **EB** is not Excel Basic, but Embedded Basic. According to Scott Ferguson (who jokingly calls himself "the mother of Visual Basic"), at the turn of the 80s and 90s, Microsoft was working on the Omega database, a project that was eventually abandoned. The Omega DBMS included the Embedded Basic (EB) engine. Development took place in a department called the "Business Programming Languages Department" which was also involved in the development of QuickBASIC. But the main resources of the department of "programming languages for business" were concentrated on the creation of a new, object-oriented, Windows-tailored BASIC-like programming language, which was called **Silver** (we have already seen this name and this product mentioned in a story by Joel Spolsky). **Bill Gates at some point sent a request to the "programming languages for business" department about whether Ruby and EB could be crossed somehow** . John Fine (the project manager at the time) was tasked with Scott Ferguson to answer this request.

A team was created that did a very difficult integration of their EB with Ruby, which they had just bought from the side:



(The two people in the photo, whose faces were painted over, were invited by the photographer from the side to improve the composition of the frame, and had nothing to do with the development of VB)

New product, born from the merger of **EB** and **Ruby**, was called **Thunder** (traces of this name survived until VB6 in the names of many internal functions (for example ThunderMsgLoop) or in the names of window classes (for example ThunderRT6FormDC). Scott Ferguson recalls that back in January 1989, John Fine wrote (probably to his superiors, possibly himself Gates) a proposal to create a programming language called "Visual BASIC". This was long before the appearance of Ruby at Microsoft. Much later, just before the release of the **Thunder product** this name was resurrected from oblivion along with many other candidate names considered as the official name of the product. And in fact, a lot of people didn't like it. Most liked the name "Thunder" (translated as "Thunder") with the then slogan "The Power to Crack **Windows**" (translated as "Power to open Windows" or perhaps "Power to make Windows tremble").

Either way, it doesn't really matter whether **EB** stands for **Excel Basic** or **Embedded Basic**. Judging from Joel's words about his contribution to bringing 4 important innovations to **EB**, who have remained in VB/VBA to this day, his vision of history and contribution to the development of the formation of VB and VBA should also be taken into account.

Obviously, there was the Silver technology and the EB engine, originally created for the Omega DBMS. EB itself, not without the participation of Joel, became part of Excel and became what is now known as VBA. EB, merging with Cooper's Ruby, influenced by Silver, became what is now known as VB. The Omega project, with EB in it, supposedly reincarnated as Access/MS Jet, of which EB is still a significant part.

The current state of affairs

Up to the latest version of the product - up to VB6, the border between **Ruby** and **EB** did not dissolve, both of these components did not merge to the degree of indistinguishability. There is still a clear boundary between them, each component does its own thing, has, so to speak, its own area of responsibility and area of tasks to be solved.

The separation is preserved even at the source level (information about the sources from which VB is compiled can be obtained from files with debug symbols). In terms of the source bundle, VB6 consists of two folders: ruby/ and vbadev/

The first contains all the **Ruby sources**, the second contains all the **EB sources** (which, as we know from Joel's article, was renamed to **VBA** at some point). For many functions and variables from Ruby, we will see the Rby prefix as a prefix, for many functions, variables and **EB structures we will see the Eb prefix**.

EB - what includes, what it does and what it is responsible for:

- **EB / VBA** as a whole is arranged as a technology that can be fastened to anything - to any external application (the term "host" / "host" is used for such an application). Attached to a host, **EB/VBA** allows you to write code that can interact with the host, manage the host object model, which ultimately makes it possible to extend the host's operation logic as you like and write automation for any host applications to which **EB is attached. /VBA**.
- **EB/VBA** does not depend on a specific host - this host depends on **EB/VBA**, and **EB/VBA** can be "screwed" to anything.
 - EB + Excel gives us VBA in Excel.
 - EB + Word gives us VBA in Word.
 - EB + AutoCAD gives us VBA in AutoCAD.
 - EB + Ruby gives us what we know as "standalone VB" (standalone VB, VB1-VB6).
 That is, **Ruby** is just a special case of a host for **EB**. **Ruby** cannot live without **EB**, but **EB** can live with

any other host.

- A code editor with syntax highlighting, autocompletion, and IntelliSense is part of **EB/VBA** . We are talking about the text field itself, in which the code is written, not including toolbars, toolbars, menus. It is for this reason that the code editor is identical in VBA and VB.
- The virtual machine that executes P-code is part of **EB/VBA** .
- The just-in-time compilation mechanism for parsing VB code and translating it into P-code is part of **EB/VBA** . Therefore, the syntax of the language is also part of **EB** , and has nothing to do with **Ruby** , and for this reason in all hosts to which **EB** is attached , the language will be the same with the same syntax rules and features (for example, the Not Not bug Arr will show up everywhere).
- The set of fundamental types and the set of operators supported by the language (And , Xor , Like , &) are part of **EB/VBA** , so no matter what host **EB** /VBA is attached to , it will all be the same everywhere. If you are a host developer and you have full control over the host sources, but EB is a black box for you, then there is no way you can get support for a new operator in the code (for example, a bitwise shift operator, which is not natively).
- The concept of error handling is part of EB/VBA, so wherever **EB/VBA** is screwed , everything will be the same everywhere.
- The concept that everything is divided into projects, and projects are made up of modules (more precisely, project items) comes from **EB/VBA** . At the same time, EB has no idea about any forms and UserControls - from the point of view of EB, there are only two types of modules (normal and object). The host can set its own subspecies for object modules. In VB6, these are classes, forms, user controls. In Excel, these are classes, user forms, sheets (worksheet) and books (workbook). Some other hosts may have their own subspecies of modules. But the general concept of projects and modules will exist in any host.
- The concept of including references to third-party type libraries is a feature of the **EB/VBA** engine , and therefore exists in VB and in all VBA uses. The very link to the TLB-shku connected to the project inside **EB** has the name **EBREF** and is described by the structure of the same name.
- All the functions known to most as “runtime functions”, such as **MsgBox** , **InputBox** , **Rnd** , **Beep** , **CreateObject** , **DateDiff** , **RGB** , **StrReverse** , **MkDir** are all part of **EB/VBA** , and therefore all these functions will necessarily be present not only in pure VB, but also in VBA: in Excel, and in Word, and in AutoCAD, and in any other host.
- The **Collection** and **ErrObject** classes are part of **EB/VBA** , and similarly can be found in pure **VB** , Microsoft Office applications, and other hosts.
- Such undocumented and therefore little known mechanisms as the expression server and the VBA event monitoring system are also part of **EB** .
- Everything that is somehow connected with the user interface and some visual manifestations during the execution of the program - in principle, is not part of **EB** . For example, the **MsgBox** function, which is part of **EB** (VBA) and shows a dialog box with a message, does not directly display the box - **EB** has no idea how the message should be shown and by what means this can be done. Instead, **EB** calls its host and calls a special callback function from the host, which is responsible for the visual part of displaying messages. Between **EB** and the host has such an interaction interface as an array of pointers to callbacks provided by the host. When the host initializes **EB** , it calls **EB 's EbInitHost** function and passes it this array. **EB** remembers a pointer to this array and, when it needs to show a message, it takes the corresponding element of the array (this address corresponds to the callback function) and calls the desired function. And it is the host code that determines how the message will be shown, whether it will be a window, what kind of window it will be, and what kind of design it will have. In this regard, **EB** is made so versatile that the host for **EB** can serve, for example, a full-screen game that uses DirectX. And in this game, calling **MsgBox** won't result in a regular windows dialog that breaks full screen mode - the callback function implemented by the game host to display messages will show a message in a game style, which will be displayed by rendering using DirectX, like all other game UI .

Ruby - what includes, what it does and what it is responsible for:

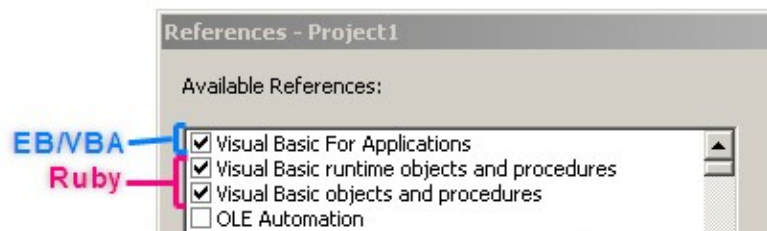
- First of all, it must be said that **Ruby** has no idea (and does not care) what language is behind the code that

manages the infrastructure - the one that Ruby provides. **In the case of VB, EB** is used in conjunction with **Ruby**, but purely hypothetical, there could be bare code in C / C ++, or some kind of scripting engine, or a visual programming system like [Scratch](#), in which the code is not written, but drawn in the form of flowcharts and diagrams. By the way, according to the original idea of Alan Cooper, this is exactly how Ruby worked: when the user drew a pair of gizmos, say, a button and a listbox, then in order to make the listbox disappear when the button was clicked, the user directly drew an arrow coming from the "click" event of the first gizmo before the "hide" method of the second gizmo. So...

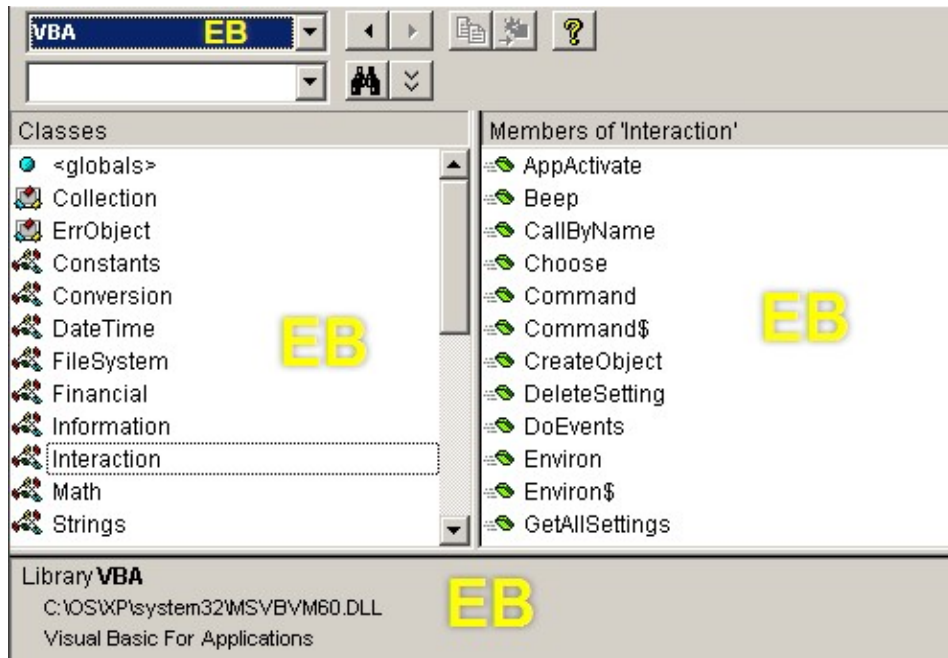
- All controls (controls) built into VB, such as CommandButton, ListBox, Drive, Circle, PictureBox, ScrollBar are all part of **Ruby**.
- The App, Clipboard, Forms collection, Printers, Screen object, **Load** and **Unload methods**, **LoadPicture** / **SavePicture method** are all part of **Ruby**. For this reason, having opened VBA in Excel, you will not be able to write App.Path or Screen.ActiveForm there - there simply will not be any App and Screen, because this is part of **Ruby**, not **EB**.
- The form engine that windows are based on is **Ruby**.
- All processing of window messages, filtering, redirection and turning window messages into events in controls is **Ruby**.
- Much of what concerns the development environment (IDE) and how to draw forms in design-time is **Ruby**. For example, the project tree, the component palette, the properties panel, the project properties dialog, the environment dialogs, the Add-in support mechanism - all this is **Ruby**.
- All the provisioning mechanisms needed to run standalone EXE files, as well as run projects compiled to form DLL/OCX files, as well as ActiveX EXE projects, are all **Ruby**. For example, any ActiveX DLL must have a **DllGetClassObject** function that returns the class factory for the requested class. **DllGetClassObject** and the class factory are all implemented in source files that are part of **Ruby**.
- Everything related to OLE, DDE and data-binding mechanisms (this is when controls on the form, for example, text fields, are bound to the fields of the recordset and change themselves when moving through the rows in the recordset, and changing the values in the controls leads to a change in the values in DB - and all this without writing additional when from the side of the VB programmer, but simply by setting up a connection between form controls and the database) - all this is **Ruby**.

This is how separation works. Assuming that you have full VB sources on hand, and you need to fix something in the behavior of a button or add a new property to forms, go to the **Ruby** sources. If you want to fix something in the built-in Collection class or have taken a swing at adding a new operator in VB, go to the **EB** sources.

The division into Ruby and EB is visible not only at the source level. If you go to Project→References, then we will see there several connected type libraries that cannot be disabled. One of them matches everything that **EB** brings, the other two correspond to everything that **Ruby** brings:

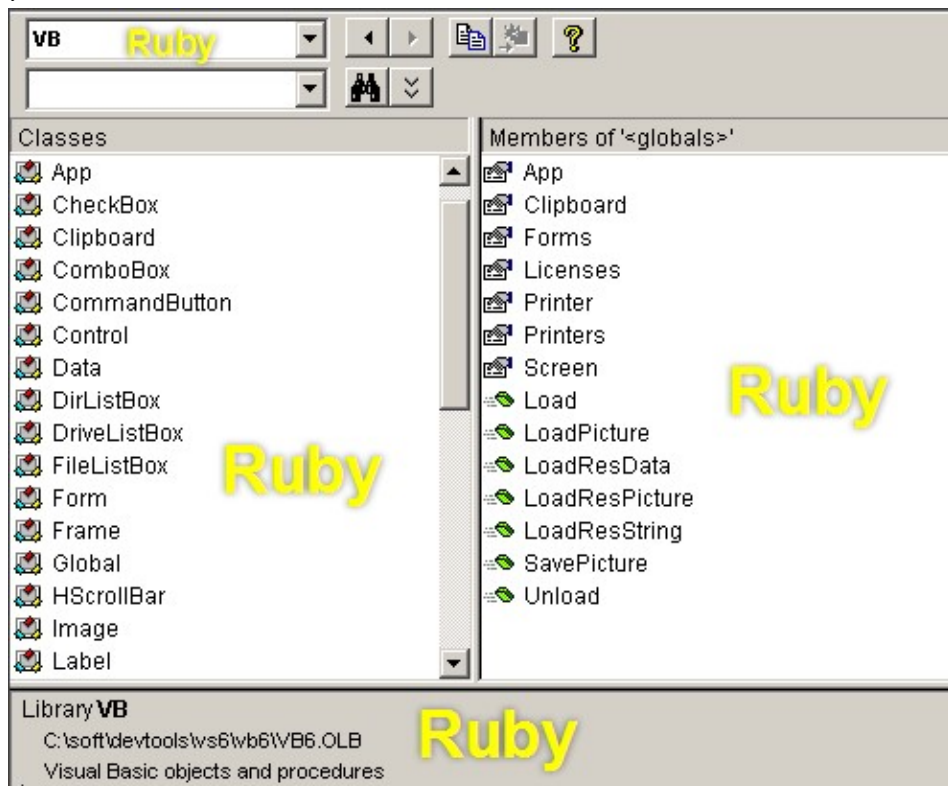


EB/VBA has its own type library, and in it you can find all the built-in runtime functions, as well as two single classes - **Collection** and **ErrObject**:



If we open Visual Basic in Excel, we will see the same type library in References and Object Browser - and all functions will be in place.

Ruby has its own type library where you can find all the built-in controls and objects like **App** , **Printers** , **Screen** :



Anatomy of VB6.EXE, VBA6.DLL and MSVBM60.DLL

Everyone knows that a compiled VB project will inevitably depend on **MSVBVM60.DLL**- as soon as this library is called (both runtime and VB virtual machine). While the project exists in the form of sources and is debugged under the IDE, its work is provided by the IDE itself, which consists mainly of two binaries:

- VB6.EXE
- VBA6.DLL (VB6.EXE depends on it)

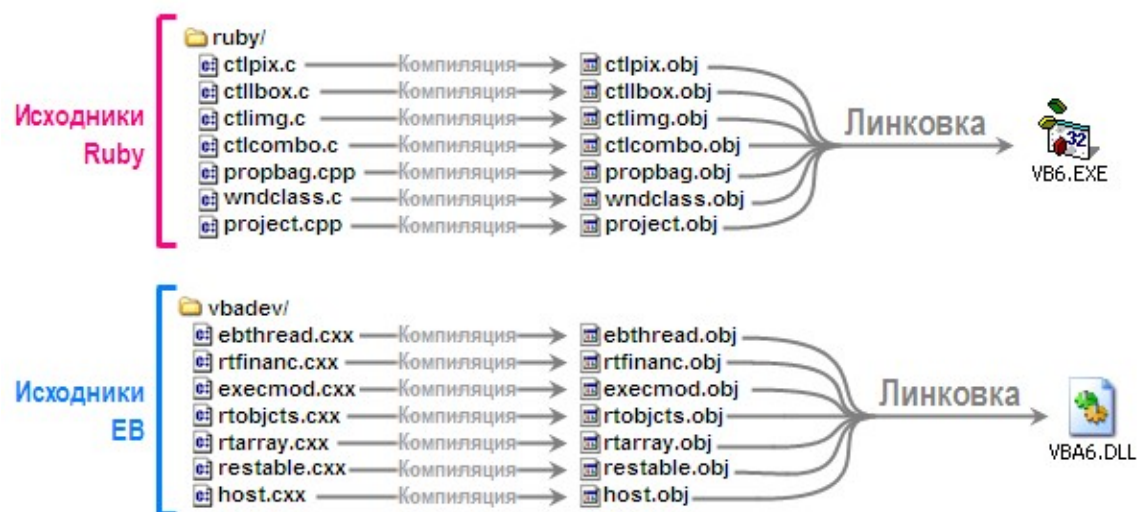
A fact that may seem surprising to some: the **inside of msvbvm60.dll** is half the **inside of VB6.EXE** and half the **inside of VBA6.DLL** .

In fact, **VB6.EXE** is the result of compiling **Ruby** sources , and **VBA6.DLL** is the result of compiling **EB** sources . The MSVBVM60.DLL **runtime** is just about half **Ruby** and half **EB** .

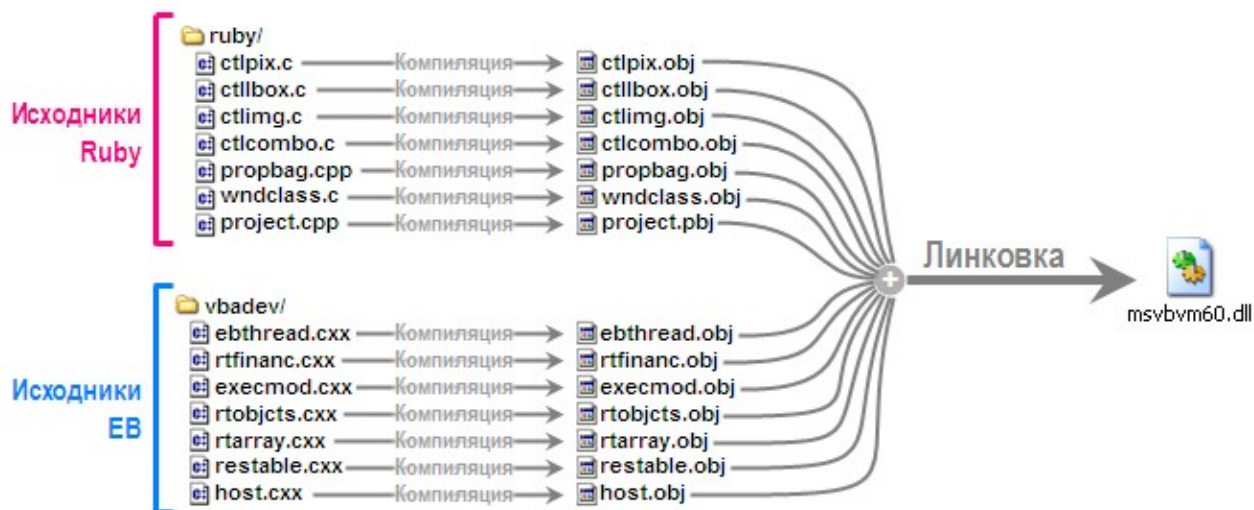
Two sets of sources (ruby/ and vbadev/), representing the two main components of this product, as a result of compilation and linking with different keys and options, they result in 3 executable files.

That is, there is no such thing as "MSVBVM60.DLL sources" and "VB IDE sources" - both are compiled and built from the same sources! But with different compilation options and switches, with different conditional compilation flags.

The process of building the IDE (VB6.EXE + VBA6.DLL) is as follows:



The process of building the MSVBVM60.DLL runtime library is almost the same: **the same** source files are compiled (with slightly different compilation switches) into object files (obj), and then everything this is linked into one final DLL file:



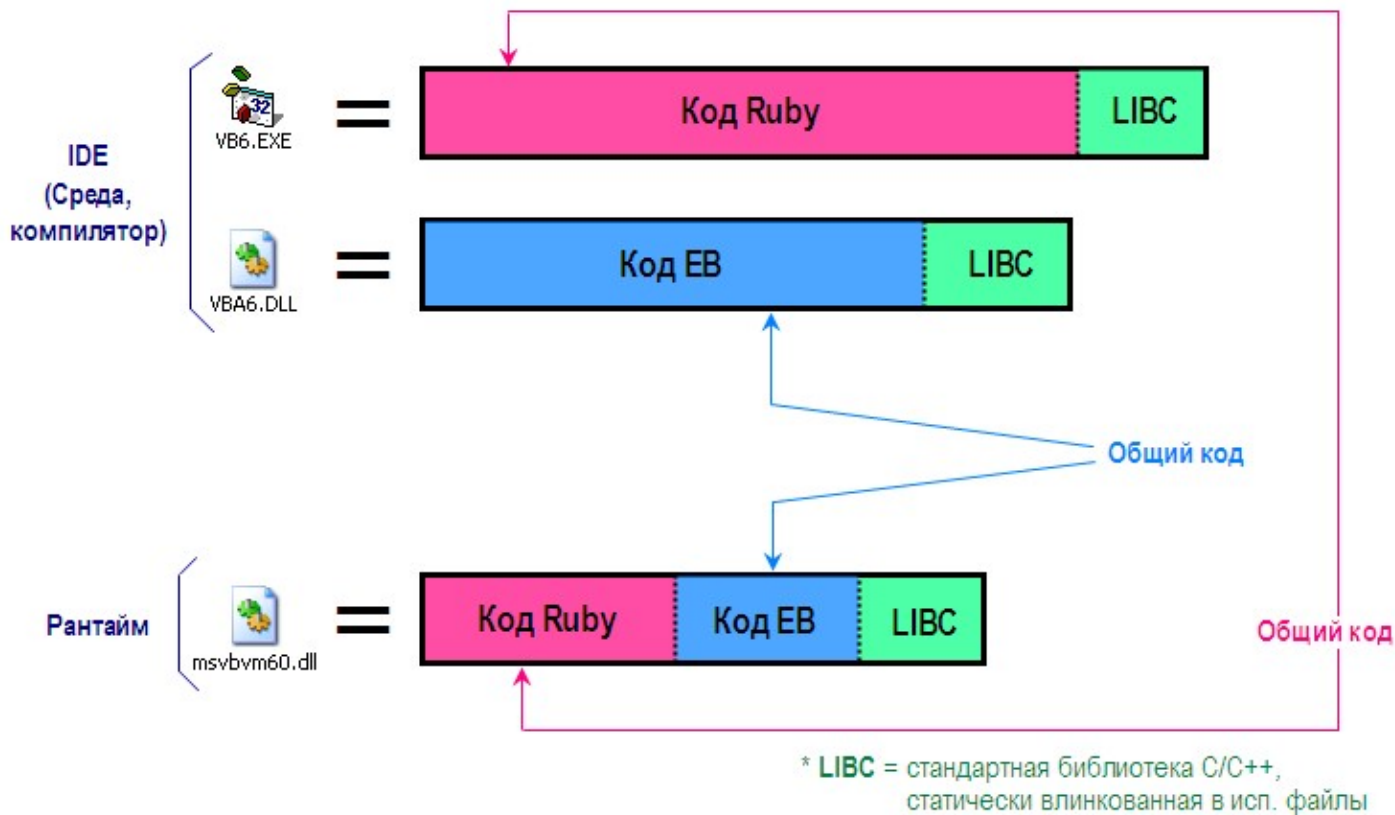
Just as the human genetic code is 98% identical to the chimpanzee genetic code, the vast majority of the binary code in **msvbvm60.dll** repeats the code from either **vb6.exe** or **vba6.dll**. Almost any procedure that is part of **msvbvm60.dll** is either part of **Ruby** or part of **EB**, and therefore can be found unchanged or slightly modified either as part of **vb6.exe** or as part of **vba6.dll**.

The opposite is not true: the total size of the **vb6.exe** code (1.80 Mb) and **vba6.dll** (1.61 MB) is larger than the size of **msvbvm60.dll** (1.32 MB).

Obviously, **Ruby** includes code that implements, for example, a form editor or a project properties dialog - all this falls into **vb6.exe**, but does not fall into **msvbvm60.dll** (it is not needed there).

Obviously, **EB** includes code that implements parsing, analysis, compilation of VB code into P code - all this gets into **vba6.dll**, but does not get into **msvbvm60.dll** (it is not needed there).

If we make some simplifications, and do not set ourselves the goal of strictly maintaining the scale, then the following illustration perfectly shows how the executable files that form the IDE (environment) and the executable file that is the runtime, in fact, consist of common code



: albeit incomplete but sufficient lists of things that are part of **Ruby** and are part of **EB** . If we simplify even more, discard the superfluous and leave only the most prominent and often remembered components of VB, then we can apply them to this illustration. Once again I want to say: this is a strong simplified image based on a highly simplified text. Its purpose is not to create a complete list of the subcomponents that make up **Ruby** and **EB** . The full list will not fit in such a small picture, the picture would have to be made at least 20 times larger. The goal is simply to look at the illustration and understand and feel the very idea, the very principle of separation. Get a rough idea of which subcomponents (responsible for certain things) are part of Ruby and which are part of EB, and how these subcomponents are distributed across executable files (VB6.EXE, VBA6.DLL and MSVBVM60.DLL). Here is this modified illustration:



In the following articles we will talk in more detail about the composition of EB and Ruby, we will consider in more detail the set of components that make up a workable VB IDE (after all, in addition to VB6.EXE and VBA6.DLL, there are also LINK.EXE, C2.EXE , MSO98RT.DLL ip).

—We separate their smiling faces from the rest of their body, Captain.

—That's right! We decapitate them.
[to come back to the beginning](#)

Hacker

Telepath



Messages: 16416

Registered: 11/13/2005 (Sun) 2:43

From: Kazakhstan, Petropavlovsk

- [Website](#)
- [ICQ](#)

Re: Ruby + EB = Visual Basic

▫ [Hacker](#) » 06/10/2019 (Mon) 14:22

I wrote this article about a year ago.

Especially for this article, two translations of other articles written by the people behind the creation of VB were made:

- Alan Cooper. [Why I'm called the "Father of Visual Basic"](#)
- Scott Ferguson. [Thunder... the birth of Visual Basic](#)

I strongly recommend that after reading this article, read these two translations in order to plunge into the descriptions of the look at the same things from the participants in the events.

—We separate their smiling faces from the rest of their body, Captain.

—That's right! We decapitate them.

[to come back to the beginning](#)

NashRus

guest



Posts: 385

Registered: 03/18/2006 (Sat) 1:16

Re: Ruby + EB = Visual Basic

▫ [NashRus](#) » 10.06.2019 (Mon) 14:56

>> and not VBA is a by-product of the creation of VB, but rather VB is a by-product of the advent of VBA.

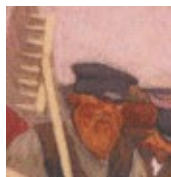
yes, it follows from MS commercial trend. The office has always been a workhorse, and still is. Many technologies

for Windows have migrated from Office. This is a good example of synergy.

[to come back to the beginning](#)

[Debugger](#)

advanced guru



Messages: 1667

Registered: 06/17/2006 (Sat) 15:11

- [ICQ](#)

[Re: Ruby + EB = Visual Basic](#)

▣ [Debugger](#) » 06/10/2019 (Mon) 16:14

This is the coolest and most unusual thing I've read about Visual Basic (and about any other programming language, I guess).

Class!

[to come back to the beginning](#)

[Teranas](#)

Experienced



Messages: 202

Registered: 12/13/2008 (Sat) 4:26

From: Novosibirsk

- [Website](#)

[Re: Ruby + EB = Visual Basic](#)

▣ [Teranas](#) » 11.06.2019 (Tue) 20:03

Great article, thanks!

Last edited by [hacker](#) on 06/11/2019 (Tue) 21:37, edited 1 time in total.

Reason: *Corrected a typo, thanks.*

With all respect, Andrew.

[to come back to the beginning](#)

[ger_kar](#)

advanced guru

**Messages:** 1956**Registered:** 05/19/2011 (Thu) 19:23**From:** Kyrgyzstan, Issyk-Kul, Karakol

[Re: Ruby + EB = Visual Basic](#)

▫ [ger_kar](#) » 11.06.2019 (Tue) 21:11

Yes, an interesting article, and leads to certain reflections. It turns out that with the development of the MS Office product line, VBA(EB) is also developing. But with the other part (Ruby) an ambush. And if for the 64-bit platform - VBA was completely adapted to itself, then with Ruby the matter is completely dull.

Fight and seek, find and hide

[to come back to the beginning](#)[Adam Smith](#)

Experienced

**Posts:** 211**Registered:** 04/25/2008 (Fri) 9:04**From:** CR. Grozny

[Re: Ruby + EB = Visual Basic](#)

▫ [Adam Smith](#) » 06/13/2019 (Thu) 17:35

ger_kar wrote: Yes, an interesting article, and leads to certain reflections. It turns out that with the development of the MS Office product line, VBA(EB) is also developing. But with the other part (Ruby) an ambush. And if for the 64-bit platform - VBA was completely adapted to itself, then with Ruby the matter is completely dull.

Apparently I've fallen behind in recent years. What is the development of VBA? Where is it being developed?

[to come back to the beginning](#)

[ger_kar](#)

advanced guru

**Messages:** 1956**Registered:** 05/19/2011 (Thu) 19:23**From:** Kyrgyzstan, Issyk-Kul, Karakol

[Re: § 10. Ruby + EB = Visual Basic](#)

▫ [ger_kar](#) » 06/13/2019 (Thu) 18:41*Adam Smith wrote:* What is the development of VBA? Where is it being developed?

Well, at least as part of MS Office products. Unfortunately, I have no information about other products that VBA was integrated into earlier, but the fact that 64-bit VBA is integrated into the 64-bit version of MS Office is a well-known fact. Also in new versions of VBA, a number of new keywords have been added. Here "The trick" recently added a [brick](#) to the piggy bank, which, in addition to everything, is universal and is designed for different versions of VBA and VB6. So in the code of this brick, you can just see some changes regarding new versions of VBA.

Fight and seek, find and hide

[to come back to the beginning](#)[Adam Smith](#)

Experienced

**Posts:** 211**Registered:** 04/25/2008 (Fri) 9:04**From:** CR. Grozny

[Re: § 10. Ruby + EB = Visual Basic](#)

▫ [Adam Smith](#) » 06/14/2019 (Fri) 6:37

Definitely kudos to the author.

Sorry, but I didn't see anything new and "64-bit" in the code.

The timer itself in the 64-bit library? So what?

Compatibility in that there are no differences from x86?
Compiling a 64 bit exe would be amazing.
I don't judge, maybe I just haven't woken up yet.
So far, even the linker from the 2010 studio has not been broken off.
[to come back to the beginning](#)

[ger_kar](#)

advanced guru



Messages: 1956

Registered: 05/19/2011 (Thu) 19:23

From: Kyrgyzstan, Issyk-Kul, Karakol

[Re: § 10. Ruby + EB = Visual Basic](#)

▢ [ger_kar](#) » 06/14/2019 (Fri) 7:26

Adam Smith wrote: Compiling a 64 bit exe would be amazing.

Well, we are talking about VBA, which does not produce any executable files at all and works exclusively with byte code.

Adam Smith wrote: Sorry, but I didn't see anything new and "64-bit" in the code.

Well, how about. Literally at the very beginning of the module, there are declarations of WinAPI functions and the following Private Declare PtrSafe Function construction immediately catches your eye, in which you can see the new PtrSafe keyword, about which the [official help](#) says the following:

The PtrSafe keyword is used in this context: Declare statement.

Declare statements with the PtrSafe keyword is the recommended syntax. Derecla statements that Ptrr include in the VBA7 development environment on32-and and after in the for 64-bit integrals or LongPtr for pointers and handles.

To ensure backwards compatibility with VBA version 6 and earlier, use the following construct:

Code: [Select all](#)

```
#If VBA7 Then
Declare PtrSafe Sub...
#Else
```

```
Declare Sub...  
#EndIf
```

When running in 64-bit versions of Office, Declare statements must include the PtrSafe keyword. The PtrSafe keyword asserts that a Declare statement is safe to run in 64-bit development environments.

Adding the PtrSafe keyword to a Declare statement only signifies that the Declare statement explicitly targets 64-bits. All data types within the scope of 4-bit values within the 4-bit values to be kept by the 64-bit integrals or retaining either bits for points and handles.

Translation of the first paragraph:

The PtrSafe keyword is used in this context:
Operator declaration.

Declaring statements with the PtrSafe keyword is the recommended syntax. Declare statements that include PtrSafe work correctly in the VBA7 development environment on both 32-bit and 64-bit platforms, only after all data types in the Declare statement (parameters and return values) that should store 64-bit quantities, updated to use LongLong for 64-bit integrals or LongPtr for pointers and descriptors.

For backward compatibility with VBA version 6 and earlier, use the following construct:

...

Walking through the documentation and going to [this page](#) , you can see that new data types have appeared in VBA, such as LongLong, LongPtr

Fight and seek, find and hide

[to come back to the beginning](#)

Hacker

Telepath

★ ★ ★ ★ ★ ★ ★ ★
★ ★ ★ ★ ★ ★ ★ ★



Messages: 16416

Registered: 11/13/2005 (Sun) 2:43

From: Kazakhstan, Petropavlovsk

- [Website](#)
- [ICQ](#)

[Re: § 10. Ruby + EB = Visual Basic](#)

▢ [Hacker](#) » 06/14/2019 (Fri) 15:22

ger_kar wrote: Well, we are talking about VBA, which does not produce any executable files at all and works exclusively with byte code.

And yet, in terms of Ruby + VBA, it is VBA that produces executable files. Whether this part of VBA was modified over time, or simply thrown out, is unknown.

—We separate their smiling faces from the rest of their body, Captain.

—That's right! We decapitate them.

[to come back to the beginning](#)

[ger_kar](#)

advanced guru



Messages: 1956

Registered: 05/19/2011 (Thu) 19:23

From: Kyrgyzstan, Issyk-Kul, Karakol

[Re: § 10. Ruby + EB = Visual Basic](#)

▢ [ger_kar](#) » 14.06.2019 (Fri) 16:52

Hacker wrote: And yet, in terms of Ruby + VBA, it is VBA that produces executable files. Whether this part of VBA was modified over time, or simply thrown out, is unknown.

It's true. But I meant exclusively pure VBA, as part of MS Office, which does not have a documented ability to create executable PE files. Maybe in its bowels it contains some code responsible for creating such files, but in fact it does not create anything. By the way, what about the ability to create (draw) forms in VBA? In theory, Ruby is responsible for this. So VBA and Ruby are also present?

Fight and seek, find and hide

[to come back to the beginning](#)

[Hacker](#)

Telepath



Messages: 16416

Registered: 11/13/2005 (Sun) 2:43

From: Kazakhstan, Petropavlovsk

- [Website](#)
- [ICQ](#)

[Re: § 10. Ruby + EB = Visual Basic](#)

▣ [Hacker](#) » 06/14/2019 (Fri) 17:12

*ger_kar wrote:*By the way, what about the ability to create (draw) forms in VBA? In theory, Ruby is responsible for this. So VBA and Ruby are also present?

Ruby is not part of the office. The ability to draw forms is implemented there independently of Ruby, the implementation of controls is completely different there (the MS Forms library is FM20.DLL for Office 2003). The set of controls is different (there is no such control as PictureBox in principle), controls do not use WinAPI windows. All controls have a different set of properties.

But the main thing: a completely different anatomy of the internal structure of these controls. After the publication of the relevant articles in the cycle about the anatomy of Ruby controls, it will be possible to return to this topic and compare.

—We separate their smiling faces from the rest of their body, Captain.

—That's right! We decapitate them.

[to come back to the beginning](#)

[ger_kar](#)

advanced guru



Messages: 1956

Registered: 05/19/2011 (Thu) 19:23

From: Kyrgyzstan, Issyk-Kul, Karakol

[Re: § 10. Ruby + EB = Visual Basic](#)

▣ [ger_kar](#) » 14.06.2019 (Fri) 19:23

*Hacker wrote:*But the main thing: a completely different anatomy of the internal structure of these controls.

Yes, the controls are completely different. In principle, MS Forms controls are quite okay, and in some ways even better than standard VB6 controls (mainly this concerns data binding), but the lack of a PictureBox and the inability to use arrays of controls greatly complicate the programming process under VBA. The inability to use arrays of controls is especially annoying. Sometimes it really infuriates when, instead of a beautiful and concise implementation, you have to fence a garden of crutches 😞

Fight and seek, find and hide
[to come back to the beginning](#)

[jungle](#)

Wikipedia



Messages: 3010

Registered: 06/03/2005 (Fri) 12:02

From: Moscow

- [Website](#)

[Re: § 10. Ruby + EB = Visual Basic](#)

▣ [jungle](#) » 06/25/2019 (Tue) 11:23

Interesting article. When will there be a sequel?
[to come back to the beginning](#)

[Hacker](#)

Telepath



Messages: 16416

Registered: 11/13/2005 (Sun) 2:43

From: Kazakhstan, Petropavlovsk

- [Website](#)
- [ICQ](#)

[Re: § 10. Ruby + EB = Visual Basic](#)

▣ [Hacker](#) » 06/25/2019 (Tue) 14:32

Soon.

—We separate their smiling faces from the rest of their body, Captain.

—That's right! We decapitate them.

[to come back to the beginning](#)

[Teranas](#)

Experienced



Messages: 202

Registered: 12/13/2008 (Sat) 4:26

From: Novosibirsk

- [Website](#)

[Re: § 10. Ruby + EB = Visual Basic](#)

▫ [Teranas](#) » 10.12.2019 (Tue) 13:19

Greetings to all!

Well, where is the promised sequel?

With all respect, Andrew.

[to come back to the beginning](#)

[jungle](#)

Wikipedia



Messages: 3010

Registered: 06/03/2005 (Fri) 12:02

From: Moscow

- [Website](#)

[Re: § 10. Ruby + EB = Visual Basic](#)

▫ [jungle](#) » 12/11/2019 (Wed) 21:59

Apparently the fuse has passed

[to come back to the beginning](#)

[Debugger](#)

advanced guru





Messages: 1667

Registered: 06/17/2006 (Sat) 15:11

- [ICQ](#)

[Re: § 10. Ruby + EB = Visual Basic](#)

▣ [debugger](#) » 12/11/2019 (Wed) 23:53

It hasn't come soon yet 😞

[to come back to the beginning](#)

Hacker

Telepath



Messages: 16416

Registered: 11/13/2005 (Sun) 2:43

From: Kazakhstan, Petropavlovsk

- [Website](#)
- [ICQ](#)

[Re: § 10. Ruby + EB = Visual Basic](#)

▣ [Hacker](#) » 12.12.2019 (Thu) 21:05

Not the fuse ended, but health ended. Seriously, the question that has occupied me in recent months, if not every day, then every other day, is the question of whether to call an ambulance again, or endure this time.

I will try to finish and publish some of the articles that are almost ready.

—We separate their smiling faces from the rest of their body, Captain.

—That's right! We decapitate them.

[to come back to the beginning](#)

Adam Smith

Experienced





Posts: 211

Registered: 04/25/2008 (Fri) 9:04

From: CR. Grozny

[Re: § 10. Ruby + EB = Visual Basic](#)

▢ [Adam Smith](#) » 12/23/2019 (Mon) 9:23

☹️ get well come on
[to come back to the beginning](#)

[Teranas](#)

Experienced



Messages: 202

Registered: 12/13/2008 (Sat) 4:26

From: Novosibirsk

- [Website](#)

[Re: § 10. Ruby + EB = Visual Basic](#)

▢ [Teranas](#) » 24.12.2019 (Tue) 15:44

It depends on what, they are spinning propaganda on the box, that their medicine is developing by leaps and bounds,
and in practice 80 percent of diseases are not treated, only the condition is maintained.
With all respect, Andrew.
[to come back to the beginning](#)

Show posts for: Sort field

[Reply](#)

Posts: 22 • Page 1 of 1

[Return to VB Internals](#)

Jump:

Who is at the conference now

Users browsing this forum: no registered users and guests: 1

- [List of forums](#)
- [Our team](#) • [Delete conference cookies](#) • Time zone: UTC + 3 hours

Powered by [phpBB](#) © 2000, 2002, 2005, 2007 phpBB Group

Time : 0.085s | 14 Queries | Gzip : On

