Visual Basic for Applications

By Contributors to Wikimedia projects

Published: 2002-02-25 · Archived: 2026-04-06 02:12:07 UTC

Visual Basic for Applications	
	
Paradigm	Multi-paradigm
Developer	Microsoft
First appeared	1993; 33 years ago
Stable release	12 712 (Office 2021)
Typing discipline	Static/Dynamic Hybrid, Strong/Weak Hybrid
OS	Microsoft Windows, macOS
License	Commercial proprietary software
Website	https://learn.microsoft.com/en-us/office/vba/library-reference/concepts/getting-started-with-vba-in-office
Influenced by	
QuickBASIC, Visual Basic	

Visual Basic for Applications (**VBA**) is an implementation of Microsoft's event-driven programming language Visual Basic 6.0 built into most desktop Microsoft Office applications. Although based on pre-.NET Visual Basic, which is no longer supported or updated by Microsoft (except under Microsoft's "It Just Works" support which is for the full lifetime of supported Windows versions, including Windows 10 and Windows 11), the VBA implementation in Office continues to be updated to support new Office features.[1][2] VBA is used for professional and end-user development due to its perceived ease-of-use, Office's vast installed userbase, and extensive legacy in business.

Visual Basic for Applications enables building user-defined functions (UDFs), automating processes and accessing Windows API and other low-level functionality through dynamic-link libraries (DLLs). It supersedes and expands on the abilities of earlier application-specific macro programming languages such as Word's WordBASIC. It can be used to control many aspects of the host application, including manipulating user interface features, such as menus and toolbars, and working with custom user forms or dialog boxes.

As its name suggests, VBA is closely related to Visual Basic and uses the Visual Basic Runtime Library. However, VBA code normally can only run within a host application, rather than as a standalone program. VBA can, however, control one application from another using OLE Automation. For example, VBA can automatically create a Microsoft Word report from Microsoft Excel data that Excel collects automatically from polled sensors. VBA can use, but not create, ActiveX/COM DLLs, and later versions add support for class modules.

VBA is built into most Microsoft Office applications, including Office for Mac OS X (except version 2008), and other Microsoft applications, including Microsoft MapPoint and Microsoft Visio. VBA is also implemented, at least partially, in applications published by companies other than Microsoft, including ArcGIS, AutoCAD, Collabora Online, CorelDraw, Kingsoft Office, LibreOffice,[3] SolidWorks,[4] WordPerfect, and UNICOM System Architect (which supports VBA 7.1).

When personal computers were initially released in the 1970s and 1980s, they typically included a version of BASIC so that customers could write their own programs. Microsoft's first products were BASIC compilers and interpreters, and the company distributed versions of BASIC with MS-DOS (versions 1.0 through 6.0) and developed follow-on products that offered more features and capabilities (QuickBASIC and BASIC Professional Development System).

In 1989, Bill Gates sketched out Microsoft's plans to use BASIC as a universal language to embellish or alter the performance of a range of software applications on microcomputers.[5] He also revealed that the installed base of active BASIC programmers was four million users, and that BASIC was used three times more frequently than any other language on PCs.

When Visual Basic was released in 1991, it seemed logical to use Visual Basic as the universal programming language for Windows applications. Until that time, each Microsoft application had its own macro language or automation technique, and the tools were largely incompatible. The first Microsoft application to debut VBA was Microsoft Excel 5.0 in 1993, based on Microsoft Visual Basic 3.0. This spurred the development of numerous custom business applications, and the decision was made to release VBA in a range of products.

Windows users learned about the changes through user groups, books, and magazines. Early computer books that introduced VBA programming skills include Reed Jacobsen's *Microsoft Excel Visual Basic for Windows 95 Step by Step* (Microsoft Press, 1995) and Michael Halvorson and Chris Kinata's *Microsoft Word 97 Visual Basic Step by Step* (Microsoft Press, 1997).

Code written in VBA is compiled[6] to Microsoft P-Code (pseudo-code), a proprietary intermediate language, which the host applications (Access, Excel, Word, Outlook, and PowerPoint) store as a separate stream in COM Structured Storage files (e.g., `.doc` or `.xls`) independent of the document streams. The intermediate code is then executed[6] by a virtual machine (hosted by the host application). Compatibility ends with Visual Basic

version 6; VBA is incompatible with Visual Basic .NET (VB.NET). VBA is proprietary to Microsoft and, apart from the COM interface, is not an open standard.

Interaction with the host application uses OLE Automation. Typically, the host application provides a type library and application programming interface (API) documentation which document how VBA programs can interact with the application. This documentation can be examined from inside the VBA development environment using its Object Browser.

Visual Basic for Applications programs which are written to use the OLE Automation interface of one application cannot be used to automate a different application, even if that application hosts the Visual Basic runtime, because the OLE Automation interfaces will be different. For example, a VBA program written to automate Microsoft Word cannot be used with a different word processor, even if that word processor hosts VBA.

Conversely, multiple applications can be automated from the one host by creating Application objects within the VBA code. References to the different libraries must be created within the VBA client before any of the methods, objects, etc. become available to use in the application. This is achieved through what is referred to as Early or Late Binding. These application objects create the OLE link to the application when they are first created. Commands to the different applications must be done explicitly through these application objects in order to work correctly.

As an example, VBA code written in Microsoft Access can establish references to the Excel, Word and Outlook libraries; this allows creating an application that – for instance – runs a query in Access, exports the results to Excel and analyzes them, and then formats the output as tables in a Word document or sends them as an Outlook email.

VBA programs can be attached to a menu button, a macro, a keyboard shortcut, or an OLE/COM event, such as the opening of a document in the application. The language provides a user interface in the form of UserForms, which can host ActiveX controls for added functionality.

Inter-process communication automation includes the Dynamic Data Exchange (DDE) and RealTimeData (RTD) which allows calling a Component Object Model (COM) automation server for dynamic or realtime financial or scientific data.[7]

As with any common programming language, VBA macros can be created with malicious intent. Using VBA, most of the security features lie in the hands of the user, not the author. The VBA host application options are accessible to the user. The user who runs any document containing VBA macros can preset the software with user preferences. End-users can protect themselves from attack by disabling macros from running in an application or by granting permission for a document to run VBA code only if they are sure that the source of the document can be trusted.

In February 2022, Microsoft announced its plan to block VBA macros in files downloaded from the Internet by default in a variety of Office apps due to their widespread use to spread malware.[8]

A risk with using VBA macros, such as in Microsoft Office applications, is exposure to viruses.[9][10] Risks stem from factors including ease of writing macros which decreases the skill required the write a malicious macro and

that typical document sharing practices allow for a virus to spread quickly.[11]

System macro virus

A system macro – one that provides a core operation – can be redefined. This allows for significant flexibility, but also is a risk that hackers can exploit to access the document and its host computer without the user's knowledge or consent. For example, a hacker could replace the built-in core functionality macros such as AutoExec, AutoNew, AutoClose, AutoOpen, AutoExit with malicious versions.[11] A malicious macro could be configured to run when the user presses a common keyboard shortcut such as `Ctrl+B` which is normally for bold font.[11]

Document-to-macro conversion

A type of macro virus that cuts and pastes the text of a document in the macro. The macro could be invoked with the Auto-open macro so that the text would be re-created when the document (empty) is opened. The user will not notice that the document is empty. The macro could also convert only some parts of the text in order to be less noticeable. Removing macros from the document manually or by using an anti-virus program could lead to a loss of content in the document. [10]:609–610

Polymorphic macros

Polymorphic viruses change their code in fundamental ways with each replication in order to avoid detection by anti-virus scanners.[12] In WordBasic (first name of the language Visual Basic), polymorphic viruses are difficult to do.
Indeed, the macro's polymorphism relies of the encryption of the document. However, the hackers have no control of the encryption key.
Furthermore, the encryption is inefficient: the encrypted macros are just in the document, so the encryption key is too and when a polymorphic macro replicates itself, the key does not change (the replication affects only the macro not the encryption).
In addition to these difficulties, a macro can not modify itself, but another macro can. WordBasic is a powerful language, it allows some operations to the macros:

- Rename the variables used in the macro(s).
- Insert random comments between the operators of its macro(s)
- Insert between the operators of its macros other, ‘do-nothing’ WordBasic operators which do not affect the execution of the virus.
- Replace some of its operators with others, equivalent ones, which perform the same function.
- Swap around any operators the order of which does not impact the result of the macro’s execution.
- Rename the macro(s) themselves to new, randomly selected names each time the virus replicates itself to a new document, with the appropriate changes in these parts of the virus body which refer to these macros.

So, in order to implement macros viruses which can change its contents, hackers have to create another macro which fulfills the task to modify the content of the virus. However, this type of macro viruses is not widespread. Indeed, hackers frequently choose to do macro viruses because they are easy and quick to implement. Making a polymorphic macro requires a lot of knowledge of the WordBasic language (it needs the advanced functionalities) and more time than a "classic" macro virus. Even if a hacker were to make a polymorphic macro, the

polymorphism needs to be done, so, the document needs to update and the update can be visible to a user.[10]:610–612

Chained macros

During replication, a macro can create do-nothing macros. But this idea can be combined with polymorphic macros, so macros are not necessarily do-nothing; each macro invokes the next one, so they can be arranged in a chain. In such a case, if they are not all removed during a disinfection, some destructive payload is activated. Such an attack can crash the winword processor with an internal error. Since Winword 6.0, the number of macros per template is limited to 150, so the attack is limited, too, but can still be very annoying. [10]:623

"Mating" macro viruses

Macro viruses can, in some cases, interact between themselves. If two viruses are executed at the same time, both of them can modify the source code of each other.
So, it results a new virus which can not be recognize by the anti-viruses software. But the result is totally random: the macro virus can be more infectious or less infectious, depending upon which part of the virus has been changed.
However, when the 'mating' is unintentional, the resulting macro virus has more chances to be less infectious. Indeed, in order to replicate itself, it has to know the commands in the source code, but, if it is changed with a random scheme, the macro can not replicate itself.
Nevertheless, it is possible to do such macros intentionally (it is different from polymorphic macros viruses which must use another macro to change their contents) in order to increase the infectivity of the two viruses.
In the example of the article,[10]:612–613 the macro virus *Colors*[13] infected a document, but another infected the user's system before : the macro virus *Concept*.
Both of these viruses use the command *AutoOpen*, so, at first, the macro virus *Colors* was detected but the command *AutoOpen* in it was the command of the macro virus *Concept*.
Moreover, when *Concept* duplicates itself, it is unencrypted, but the command in the virus *Colors* was encrypted (Colors encrypt its commands).
So, replication of the macro virus *Concept* results in the hybridation of this macro virus (which had infected the user's system first) and *Colors*.
The "hybrid" could replicate itself only if *AutoOpen* were not executed; indeed this command comes from *Concept*, but the body of the hybrid is *Colors*, so that create some conflicts.
This example shows the potential of mating macro viruses: if a couple of mating macro viruses is created, it will make it more difficult to detect both macro viruses (in this hypothesis, there are only two viruses which mate) by the virus-specific scanners and may reinforce the virility of the viruses.
Fortunately, this type of macro virus is rare (more than the polymorphic macro viruses, one may not even exist), indeed, creating two (or more) which can interact with each other and not reduce the virility (rather reinforce it) is complicated.

Macro virus mutators

Macros can be designed to modify existing Macros, such a tool would allow one to create a new virus by modifying an existing one. This is difficult to do with executable files. But it is very simple for macro viruses,

since the source code of macros are always available. In a similar manner to polymorphic macros, a macro can perform modifications to all macros present in the document. Considering this, just a few modifications could create a macro virus mutator, and thereby quickly create several thousands of known viruses. [10]:613–614

Parasitic macro viruses

Most macros viruses are stand-alone; they do not depend on other macros (for the infectious part of the virus, not for the replication for some viruses), but some macros viruses do. They are called parasitic macros.[10]:614–615 When launched, they check other macros (viruses or not), and append their contents to them. In this way, all of the macros became viruses. But, this type of macro can not be spread as quickly as stand-alone macros. Indeed, it depends on other macros, so, without them, the virus can not be spread. So, parasitic macros often are hybrid: they are stand alone and they can infect other macros. This kind of macro virus poses real problems to the virus-specific anti-virus; in fact, they change the content of other viruses, so that accurate detection is not possible.

Suboptimal anti-virus

[edit]

There are different types of anti-virus (or scanners), one is the heuristic analysis anti-virus which interprets or emulates macros.
Indeed, to examine all branches of macros require a NP-complete complexity[10]:605 (using backtracking), so in this case, the analysis of one document (which contains macros) would take too much time. Interpreting or emulating a macro would lead to either false positive errors or in macro viruses not detected.

Another type of anti-virus, the integrity checker anti-virus, in some cases, does not work: it only checks documents with extensions DOT[14] or DOC (indeed, some anti-virus producers suggest to their users), but Word documents can reside in others extensions than those two, and the content of the document tends to change often. [10]:605
So, like the heuristic analysis, this can lead to false positives errors, due to the fact that this type of anti-virus checks the whole document.

The last type of anti-virus seen will be the virus-specific scanner.[10]:608 It searches the signature of viruses, so, the type of anti-virus is weaker than the previous ones.
Indeed, the viruses detected by virus-specific scanners are just the ones known by the software producers (so, more updates are needed than in other types of scanners). Moreover, this type of anti-virus is weak against morphing viruses (cf.section above). If a macro virus change its content (so, its signature), it cannot be detected any more by the virus-specific scanners, even if it is the same virus doing the same actions. Its signature does not match the one declared in the virus scanner.

Additional to the responsibility of the anti-virus is the user's responsibility: if a potential macro virus is detected, the user can choose what to do with it: ignore it, quarantine it or destroy it, but the last option is the most dangerous.
The anti-virus can activate some destructive macro viruses which destroy some data when they are deleted by the anti-virus.

So, both virus scanners and users are responsible for the security and the integrity of the documents/computer. Moreover, even if the anti-virus is not optimal in the virus detection, most macro viruses are detected and the progression in virus detection improves but with creation of new macro viruses.

- VBA was first launched with MS Excel 5.0 in 1993. It became an instant success among developers to create corporate solutions using Excel. Inclusion of VBA with Microsoft Project, Access and Word replacing Access BASIC and WordBASIC respectively made it more popular.
- VBA 4.0 is the next famous release with a totally upgraded version compared to previous one. Released in 1996, it is written in C++ and became an object oriented language.
- VBA 5.0 was launched in 1997 along with all of MS Office 97 products. The only exception for this was Outlook 97 which used VBScript.
- VBA 6.0 and VBA 6.1 were launched in 1999, notably with support for COM add-ins in Office 2000. VBA 6.2 was released alongside Office 2000 SR-1.
- VBA 6.3 was released after Office XP, VBA 6.4 followed Office 2003 and VBA 6.5 was released with Office 2007.
- Office 2010 includes VBA 7.0. There are no new features in VBA 7 for developers compared to VBA 6.5 except for 64-bit support. However, after VBA 6.5/Office 2007, Microsoft stopped licensing VBA for other applications.
- Office 2013, Office 2016, Office 2019, Office 2021 and Office 2024 include VBA 7.1.

As of July 1, 2007, Microsoft no longer offers VBA distribution licenses to new customers. Microsoft intended to add .NET-based languages to the current version of VBA ever since the release of the .NET Framework,[15] of which versions 1.0 and 1.1 included a scripting runtime technology named *Script for the .NET Framework*.[16] Visual Studio .NET 2002 and 2003 SDK contained a separate scripting IDE called *Visual Studio for Applications* (VSA) that supported VB.NET.[17][18][19] One of its significant features was that the interfaces to the technology were available via Active Scripting (VBScript and JScript), allowing even .NET-unaware applications to be scripted via .NET languages. However, VSA was deprecated in version 2.0 of the .NET Framework,[19] leaving no clear upgrade path for applications desiring Active Scripting support (although "scripts" can be created in C#, VBScript, and other .NET languages, which can be compiled and executed at run-time via libraries installed as part of the standard .NET runtime).

Microsoft dropped VBA support for Microsoft Office 2008 for Mac.[20][21] VBA was restored in Microsoft Office for Mac 2011. Microsoft said that it has no plan to remove VBA from the Windows version of Office.[22][23]

With Office 2010, Microsoft introduced VBA7, which contains a true pointer data type: LongPtr. This allows referencing 64-bit address space. The 64-bit install of Office 2010 does not support common controls of MSComCtl (TabStrip, Toolbar, StatusBar, ProgressBar, TreeView, ListViews, ImageList, Slider, ImageComboBox) or MSComCt2 (Animation, UpDown, MonthView, DateTimePicker, FlatScrollBar), so legacy 32-bit code ported to 64-bit VBA code that depends on these common controls will not function. This did not affect the 32-bit version Office 2010.[24] Microsoft eventually released a 64-bit version of MSComCtl with the July 27th, 2017 update to Office 2016.[25]

- Visual Studio Tools for Applications

- Visual Studio Tools for Office
- Microsoft Visual Studio
- Microsoft FrontPage
- OpenOffice Basic
- LotusScript
- Microsoft Power Fx

1. ^ *"Compatibility Between the 32-bit and 64-bit Versions of Office 2010"*. *msdn.microsoft.com.*
2. ^ *o365devx. "What's new for VBA in Office 2019". docs.microsoft.com. Retrieved 2022-05-02.* {{cite web}} : CS1 maint: numeric names: authors list (link)
3. ^ *"Support for VBA Macros". The Document Foundation - LibreOffice. Retrieved 3 January 2023.*
4. ^ *"2016 SolidWorks Help – VBA". help.solidworks.com. Retrieved 2016-07-25.*
5. ^ *Gates, Bill; Halvorson, Michael; Rygmyr, David (1989). Learn BASIC Now. Redmond, WA: Microsoft Press. pp. ix–x.*
6. ^ Jump up to: ***a*** ***b*** *"ACC: Visual/Access Basic Is Both a Compiler and an Interpreter". Microsoft. 2012. Archived from the original on 2012-10-21.*
7. ^ *"How to set up and use the RTD function in Excel". msdn.microsoft.com.*
8. ^ *"Microsoft to Block Office VBA Macros by Default". The Verge. 7 February 2022. Retrieved 2022-09-26.*
9. ^ *Vesselin Bontchev. "Macro Virus Identification Problems". macros viruses.* {{cite web}} : CS1 maint: deprecated archival service (link)
10. ^ Jump up to: ***a*** ***b*** ***c*** ***d*** ***e*** ***f*** ***g*** ***h*** ***i*** ***j*** *Vesselin Bontchev (1996). "Possible macro virus attacks and how to prevent them". Virus, Macros, Safety of Macros.* ***15*** *(7): 595. doi:10.1016/S0167-4048(97)88131-X.*
11. ^ Jump up to: ***a*** ***b*** ***c*** *Paul Docherty; Peter Simpson (1999). "Macro attacks: What next after Melissa?". Viruses, Safety of Macros.* ***18*** *(5): 391–395. doi:10.1016/S0167-4048(99)80084-4.*
12. ^ Polymorphics macros
13. ^ *"Macro Virus Colors". Archived from the original on 2012-11-22. Retrieved 2012-12-15.*
14. ^ DOT extension
15. ^ *"Visual Studio for Applications". Archived from the original on 2007-12-17.*
16. ^ *"Introducing Visual Studio for Applications". msdn.microsoft.com.*
17. ^ *"Script Happens .NET". msdn.microsoft.com.*
18. ^ *"Microsoft Takes Wraps Off VSA Development Technology". Archived from the original on 2007-12-17.*
19. ^ Jump up to: ***a*** ***b*** *"VSA scripting in .NET". Archived from the original on 2007-02-11.*
20. ^ *"WWDC: Microsoft updates Universal status of Mac apps". Macworld. 2006-08-07. Archived from the original on 2008-07-19. Retrieved 2007-05-25.*
21. ^ *"What is Microsoft Office and Office 365 – FAQs".*
22. ^ *"The Reports of VBA's Demise Have Been Greatly Exaggerated".*
23. ^ *"Clarification on VBA Support". Archived from the original on 2008-04-11.*
24. ^ *"Compatibility Between the 32-bit and 64-bit Versions of Office 2010". msdn.microsoft.com.*
25. ^ *"Release notes for Monthly Channel releases in 2017". learn.microsoft.com. Retrieved 2022-11-13.*

Source: https://en.wikipedia.org/wiki/Visual_Basic_for_Applications