Picking up a VB6 project for a non-VB6 guy [closed]

Asked 16 years, 1 month ago Modified 16 years, 1 month ago

Viewed 1k times

5

Closed. This question is opinion-based. It is not currently accepting answers.

Want to improve this question? Update the question so it can be answered with facts and citations by editing this post.

Closed 6 years ago.

Improve this question

I have been given the task of modifying a VB6 project. Nothing incredibly serious, adding a couple forms and fixing a few bugs for the most part. The project uses SQL Server (if that is of any relevance).

My background in programming has been VB/C# .NET, PHP, C++ and mostly MySQL although I have used SQL Server on a much smaller scale. What kind of advice can anyone give me or resources for VB6 programming. It's been a few years since I've done any VB .NET work and while I can read the VB6 code and understand what is going on, I'm not sure how well I'm going to be able to start writing and/or modifying without a chance of breaking anything.

What kind of advice can others offer? Any resources and/or stories would be great. Feel free to contribute things you may feel relevant yet I neglected to mention.

Thanks!

.net sql-server vb6

Share

Improve this question

Follow

asked Nov 11, 2008 at 21:08

community wiki
Ryan Rodemoyer

I really want to answer this because I spent the first 6 years of my career coding in VB before .NET came out. It's been so long though since I did any work in VB6 that I can't remember anything useful. I feel for you though and hope you don't have to work in it for very long. – Tad Donaghe Nov 11, 2008 at 21:25

Great book on how to do vb6 well: amazon.com/…
– adolf garlic Apr 7, 2010 at 19:08

10 Answers

Sorted by: Highest score (default)

14

It is probably already there, but make sure **Option Explicit** is at the top of all of the files. It forces variable declaration and reduces the chance of a typo inadvertently creating a variable on the fly.

Share Improve this answer Follow

answered Nov 11, 2008 at 21:29

community wiki
jac

I'm familiar with Option Explicit from VBScript and VB .NET. However, I've already looked through some of the source and they might as well as thrown variable scope out the window. – Ryan Rodemoyer Nov 11, 2008 at 22:32

1 Yours is the only advise that can be taken without a grain of salt here. The rest ranges from baloney to pure toxic. :-/ – Konrad Rudolph Nov 11, 2008 at 23:01

Well, depending on how bad the code base is, turning on Option Explicit might instantly mean a huge amount of work just to compile. In fact, the only people who didn't put that in in the first place are the kind of people who created all the VB messes. – MusiGenesis Nov 11, 2008 at 23:46

1 It's funny that all the world is crazy about languages that let you use variables without declaring them .. – Eduardo Molteni Nov 12, 2008 at 16:42

@Eduardo I agree. And duck typing too. After years of pleading Option Strict went into VB to allow us to avoid evil type coercion (duck typing). Apparently Joel was to blame in the first place joelonsoftware.com/items/2009/03/09.html – MarkJ May 8, 2009 at 12:35

14

VB6 has a broken type system - in fact it has two type systems that are not totally compatible. Architecturally, VB4-6 are a fairly thin wrapper over COM, and use COM's type system. Previous versions of VB had their own type system, which was sort of a cross between traditional BASIC and C. The two can't be freely mixed and matched.

There is no inheritance and no real exception handling, so expect to write a lot of boilerplate code for error handling and basic form functions repeated over and over. This is where VB gets its reputation for cut-and-paste software engineering.

Some of the types you can declare in VB are not 'automation compatible', which means that they cannot be serialised over a COM boundary or stored in a Variant (more on those later). These types date back to pre-VB4-6 days when VB did not support COM. One of these types is a record (like a c struct) type whose name escapes me, so an expandible collection or associative array of structs (which is a natural thing that one might want to do) is not possible. Defining a class (see below) is possible but somewhat clumsy.

VB does not support multi-threading, and the workarounds for this have many gotchas. The first of these will potentially bite you if you are doing 3-tier development using MTS or COM+. Creating VB module actually creates a singleton COM object behind the scenes. This will live in its own single-threaded apartment. Apartments are essentially lightweight running COM servers with a serialisation/deserialisation mechanism (called a Message Pump) where calls get serialised and queued for a single thread. If you do the (on the surface of it) sensible thing and try to modularise your code you will create hot-spots in your middle-tier. The solution: more cut and paste.

The second major gotcha is that COM's garbage collection is rather basic - a simple reference counting mechanism. This means that COM components that crash or for some reason don't tidy up after themselves leak memory. Remember how VB6 is just a thin layer over COM? This tight coupling means that you have to be fairly careful when using UI management code that holds references to controls, OLE automation on external applications (e.g. Excel) or anything else that sets up references. VB is quite good at doing hidden stuff behind your back and not knowing when to clean up after itself. It is also a source of circular references generated behind the scenes. Get this wrong and it will leak resources - you need to be careful about this.

The other major gotcha is Variants. The best description I've seen for a Variant type is a 'Spreadsheet Cell'. These can cause quite a lot of mischief, especially Variant Arrays. Many API's will only work with variants or have random parts that use variants (Excel does this a lot) so you won't always be able to avoid them. If you're serialising stuff over COM boundaries (for example multiple disconnected Recordsets) you will soon learn to loathe Variant Arrays.

You will probably find that VB's type system is so broken that the easiest way to maintain a non-trivially complex data structure is to do a library that encodes it in a string and essentially serialises and de-serialises it. Complex data structures in a VB6 app are almost a non-starter.

Finally, after using the GUI toolkit for VB6, you will come to learn just how much the team that did WinForms learned from the VB6 team's mistakes.

Although it was promoted as being simple and easy to use, building a non-trivial applicaiton in VB without making a mess was a lot harder than it looked due to all of the architectural flaws and little gotchas. It's quite a good example of Spolsky's Law of Leaky Abstractions..

Appleman's Developing COM/ActiveX Components in Visual BASIC 6 has quite a good treatment of the ins and outs of COM's interaction with VB6. He also did quite a good one on Win32 and VB programming..

P.S. (thanks for reminding me Daok), if you find someone's been using `On Error Resume Next`, you have my permission to head-butt them.

Share Improve this answer Follow

edited Nov 12, 2008 at 16:42

community wiki
8 revs
ConcernedOfTunbridgeWells

All that said and me and fellow programmers did some pretty amazing stuff with VB6! :) But thank Buddha for .net!
– Tad Donaghe Nov 11, 2008 at 21:48

Amen to that. When people asked me about .Net I used to say 'Imagine a version of VB that doesn't suck'
– ConcernedOfTunbridgeWells Nov 11, 2008 at 21:54

Hey, let's not forget that many of the good parts of Visual Studio first appeared in VB. – MusiGenesis Nov 11, 2008 at 22:16

@Terry Just because something has been used to make something good, doesn't make the tool good. I mean, they didn't make the pyramids with cranes and bulldozers. It takes a good programmer to make something good, a good language helps. I am not saying anything good or bad about VB just the reasoning – Vincent McNabb Nov 12, 2008 at 8:01

9

Since I cut my teeth on Visual Basic (the resume says VB3-6 10+ years), I'll try actually answering your question with something other than "On Error Resume Next" sucks (which it does, but you'll find it all over the place, or you won't, which is worse). Here are some things you can do:

1. When you take over a VB project, the hardest thing is usually just getting it to compile on your machine. This is often due to broken references to things that were installed on the original developer's machine but not on yours (a.k.a. "DLL Hell"). You can only hope the installers still exist somewhere, or if not you can always resort to dropping DLLs and OCXs into the system directory (remember to register them). Sometimes the DLL Hell comes from broken internal references. VB developers usually loved to divide applications up into as many different component projects as possible. Often many or all of your compatibility problems vanish when you just combine

everything into the one damn project it should have been in the first place.

2. Once everything compiles, you're 99% there. Most of the common problems mentioned in posts here are the kind of problems that manifest themselves in an application immediately. If the app you're inheriting is a simple, reliable app that's been humming along quietly all these years (it happened in VB), you're probably going to be fine.
3. Once you have the app compiled and working from the IDE (I think that's what it was called back then), you can start experimenting with small changes here and there.
4. **Post your problems here**. It's fun to be reminded of the past.

Regarding Visual Basic in general, I would say don't believe the anti-hype. It was a language that allowed good code or bad code, much like any other. And remember that all the awful VB programmers are now awful C# programmers.

Share Improve this answer Follow

edited Nov 12, 2008 at 0:27

community wiki
2 revs
MusiGenesis

1 I don't entirely agree with this. VB is the only language I've ever worked with (with the possible exception of SQL) that had architectural limitations - largely due to its tight coupling with COM - that made it hard to write good code in. – ConcernedOfTunbridgeWells Nov 12, 2008 at 8:09

VB 3 certainly had some architectural limitations (no classes - I have no idea how I did anything without classes). – MusiGenesis Nov 12, 2008 at 14:27

+1 It's nice to see realistic comments about VB. It never was the "best," but it's also far from the worst. – JeffK Nov 26, 2008 at 19:41

1 I'd say post your problems on the Microsoft VB6 group, still heaving with *active* VB6 developers groups.google.co.uk/group/… – MarkJ Jan 26, 2009 at 18:37

@MarkJ: I think you meant "still DRY-heaving with active VB6 developers". :) – MusiGenesis Jan 27, 2009 at 1:45

7

Do not use "**On error resume next**" please.

Share Improve this answer Follow

edited Nov 12, 2008 at 1:43

answered Nov 11, 2008 at 21:11

Patrick Desjardins
141k ● 89 ● 294 ● 346

another +1 - worst "language feature" ever... – Erik Forbes Nov 11, 2008 at 21:50

or ("shudder") the "Stop" statement. – dkretz Nov 11, 2008 at 22:18

I loved using "End" in a pinch. – MusiGenesis Nov 11, 2008 at 22:21

6

If your application uses ActiveX DLL projects then you make sure this is the situation to minimize DLL Hell

- Set your DLL to BinaryCompatibility.
- Make sure you have a Compatible directory.
- Put the DLL of the Last version in there.
- Point the Binary Compatibility to that DLL.
- Make sure your EXE and DLLs compiles to it's project directory.
- Run the EXE from it's project directory when you test.That way it will use the DLL that you compiled.
- You need to write a utility so that you can compile every project separately in the correct order. The VB6 Compilier can be run from the command line.
- Test your setup using Virtual PC or another computer.

Despite the lack of inheritance you will find that VB6 is able to implement many common object oriented design patterns. If you look in Design Patterns, Elements of Reusable Object-Oriented Software you will see that most of the patterns involve implementing interface rather than inheriting behavior. They talk about this issue starting on page 16.

Strongly typed collections are not straight forward to implement, You write the collection with everything including a readonly Item property. When you are finished you will need to hit{F2} and bring up the object browser. Find the collection class you created and right click Item.

You will need to

- Select Properties
- Click the Advanced Button
- Change the Procedure ID to Default

Then Item will become the default property and the Strongly type collection will act as expected.

To enable the use of the For Each on the collection class you will need to add this

```
Public Property Get NewEnum() As IUnknown
    Set NewEnum = mCol.[_NewEnum]
End Property
```

With mCol being the name of the private Collection variable

Again use the Object Browser and right click on NewEnum.

You will need to

- Select Properties
- Click the Advanced Button
- Change the Procedure ID to -4

Remember that Integer is 16-bit and Long is 32-bit. I recommend declaring most integers variables as Long. Back in the day it mattered for speed and memory footprint but with today's computers it better just to use Long and not worry about exceeding the limits.

Like suggested elsewhere use Option Explicit.

Visual BASIC 6 is very good at implicit type conversion. You have the Cxxx series of conversion functions if you want to be sure.

Variant is better than .NET's object at dealing with a diverse range of Object Types including Classes. You may find it useful if you need to make a custom form dealing with the database and the user can pick which table to use for that form. Using Variant makes it easier to deal with the fact that fields are of different types for different tables.

Visual Basic 6 can be used to make multi-tier applications. Forms can implement interfaces like Classes can.

Remember that ActiveX controls run when you compile and edit a form. This may cause all sorts of strangeness if you are not aware of it. This is particularly problematic if you have your own ActiveX Controls.

Share Improve this answer Follow

answered Nov 12, 2008 at 13:32

community wiki
RS Conley

2

Rather use `on error resume next` than `on error goto x`, but ALWAYS catch errors right away and then reset the error handling.

Example:

```
...
On Error Resume Next
oDbConn.Open sDbConnString
Select Case Err.Number
  Case &H80004005
    MsgBox "Cannot connect to SQL-server, check your s
    frmSettings.Show
    Exit Sub
  Case Else
    ShowErrorAndQuit Err
End Select
On Error Goto 0
...
```

Share Improve this answer Follow

answered Nov 11, 2008 at 21:48

community wiki

Svante Svenson

Ah, the under-used pseudo-try-catch of Visual Basic. Beautiful. – MusiGenesis Nov 11, 2008 at 22:18

For one line of code like this simple example, it **looks like** try-catch, but it actually requires a separate `Select Case Err.Number` check after **every** line of code that can potentially throw an error! For real world code, it is better to use `On Error Goto X` And have the `Select Case Err.Number` check at the `X` tag. If you want to be able to continue where you left off, run `Resume Next` after the error handler code. There are no simple mechanism in VB6 that can fully simulate the try-catch way of handling exceptions. – awe Oct 7, 2009 at 9:11

2

The general wisdom when declaring objects was to avoid using "Dim myObj As New MyClass". Instead use "Dim myObj As MyClass" and explicitly instantiate the object when needed with "Set myObj = New MyClass". The former method does not instantiate the object when the "Dim" is encountered, but will automatically do it the first time one of the object's properties or methods is referenced. This required some additional overhead by the runtime to do, since it must check if the object exists before every property/method reference. Also, there were some other quirky side-effects of that method of object declaration. For instance, if myObj is set to Nothing, and the program encounters another reference to it, a new instance is automatically created. Chances are that if this situation occurs, something is wrong with your program, but there will be no error generated telling you that your object doesn't exist, which can lead to some hard to track down bugs. Generally, it is better to explicitly create and destroy the object instance yourself using the "Set = New" syntax. Proponents of the "Dim As New" syntax usually cite better readability, but it's potential pitfalls can catch you sometimes.

Share Improve this answer Follow

edited Nov 12, 2008 at 16:34

community wiki
2 revs
E Brown

1

The people who posted here and said that "On Error Resume Next" is the worst language feature ever are dead wrong.

The worst language feature ever was "**On Error Resume**". Trust me on this one.

For the VB-impaired, On Error Resume Next means "something went wrong, on to the next line". On Error Resume means "something went wrong, better try it again and again and again and again and ... ".

Share Improve this answer Follow

answered Nov 11, 2008 at 22:14

community wiki
MusiGenesis

msdn.microsoft.com/en-us/library/aa266173(VS.60).aspx On Error Resume Next is not wrong. I'll edit your post or you do it? – Patrick Desjardins Nov 11, 2008 at 22:18

Did you read my whole answer? – MusiGenesis Nov 11, 2008 at 22:24

I think you miss understand the On Error Resume Next but well. If something is wrong, it goes to the next line without warning. On error resume require something after the resume... "goto" or "resume" ... not nothing... like you say – Patrick Desjardins Nov 12, 2008 at 0:27

I see what you're saying. When I said "something went wrong, on to the next line", I wasn't implying that you got notified of this in any way - you didn't. – MusiGenesis Nov 12, 2008 at 2:18

And "On Error Resume" was less famous than "On Error Resume Next", but in fact you could use Resume without specifying any argument, which would cause VB to re-execute the line that caused the error in the first place. – MusiGenesis Nov 12, 2008 at 2:19

1

Personally, I like `On Error goto ErrorHandler`, with an error handler at the bottom of each function.

Remember to never include parentheses when calling a method unless you're going to look at its return value. It's easy to get away with this on one-argument methods, but sometimes it bites you since passing (variable) is **not** the same as passing variable.

The people writing VB weren't quite sure whether they prefered 0-indexed arrays or 1-indexed arrays. You can use `UBound`/`LBound` to check where an array starts and ends.

Do not use "textcomparison" to convert case. Comparing strings as text does more than merely ignore case. Just convert one or both strings to uppercase.

Projects->Properties->Remove information about unused ActiveX Controls must be off if you're calling them at runtime.

Tools->Editor->Auto Syntax Check. If you're not used to it, maybe turn it off.

Tools->Editor->Require Variable Declaration. On.

Tools->Environment->When a program starts->Save Changes. If this is off, your program won't be saved even if you run it.

Use Run->Start With Full Compile, Not Run->Start, to run your code.

Share Improve this answer Follow

answered Nov 11, 2008 at 22:16

community wiki
Brian

Note: the `Tools->Editor->Auto Syntax Check` just specifies if you should get a message box with the error immediately when you move to another line. This is very annoying if you write something temporary, and want to copy code from somewhere else. With this option turned off, the IDE still tells you immediately that something's wrong by marking it as red, but spares you for the annoying message box. So yes - you should turn this off! – awe Oct 7, 2009 at 9:19

Note2: `Run->Start With Full Compile` will compile your entire code *before* it runs your program. `Run->Start` will just run your code, and compile in runtime what's needed, which can hide potentially compile errors in parts of the code that you are not running. So yes again on this (just to explain why...) – awe Oct 7, 2009 at 9:23

0

Do not use `On Error Resume Next` unless really necessary (there are some unique cases, like final cleanup after error processing or VB Collection element existence check, but usually you don't need it inside function body).

Implement proper error handling, do not use more than one exit point, do not use If-Then one-liners, use Case Else block, do not use Subs - something like next very simple function skeletion:

```
Private Function MyFunc() As Boolean
    On Error Goto ErrHandler

    ''some code here
    If SomeBadExitConditionIsSet Then
        GoTo FuncExit
    End If

    ''some more code here
    MyFunc = True

FuncExit:
    ''kill error handling to avoid error cycling
    On Error Resume Next
    ''cleanup code here - you need to check Err.Number
operations!
    ''single exit point
    Exit Function

ErrHandler:
    ''you can insert local specific error processing h
Err.Number etc
    ''there is possibility to raise errors to caller f
    ''but in this function we do not do that
    Select Case MyGlobalErrHandler(Err.Number, Err.Des
        Case eRetry
            Resume    ''this is useful while debugging
Next Statement"
        Case eIgnore
            Resume Next
        Case eCancel
            Resume FuncExit
        Case Else
            Resume FuncExit
    Select End
End Function
```

Share Improve this answer Follow

answered Nov 11, 2008 at 22:28

community wiki
Arvo