A Publication of Mobilize.Net

Get Off VB6

GUIDE TO **VB6** END-OF-LIFE

WHAT YOU NEED TO KNOW

As of June 2014, Microsoft declined to open-source VB6

- That means the future for VB6 is more bleak than ever before.
- Hackers can exploit every security vulnerability.
- PLUS, if you are still using VB6, you are out of compliance with:
  - HIPAA
  - PCI
  - SOX
  - And lots more.

What does EOVS mean?

- End of vendor support.
- No…
  - Updates
  - Bug fixes
  - Patches
  - Security holes plugged.
  - Ever.
  - Again.

How big a problem is it?

- IDC estimates there are more than 14 billion lines of VB6 code.
- VB6 doesn't support any modern platforms like web.
- Finding VB6 developers is hard – finding C# and web devs is easy.
- VB6 is forcing enterprises to continue to run Windows XP – which introduces a whole host of issues.

Why should I care?

- VB6 was a great idea *in its time*.

- That time has passed.
- VB6 is procedural, not object oriented.
- VB6 wraps the Win32 API; it has nothing to do with .NET
- All Microsoft OS versions since XP use .NET at their core
- Ok, Windows 8 uses something different. But it's still not Win32.

Why should I care?

- VB6 limits you to decades-old technology. That means no SaaS, no SOA, no modern web.

- And then there's the security issues.
- VB6 hasn't been updated in years and doesn't have patches to address all of the recent hacking.
- Only computers completely isolated from the web are safe.
- That means you can't do even the most basic business tasks.
- Plus, VB6 is so outdated. It's impossible for you to add functionality that you need.
- Kiss BYOD goodbye.

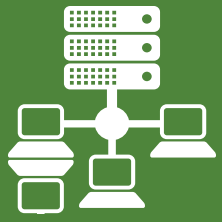
Custom LOB apps tether you to XP

Over the years, you've developed custom software, crucial to running your business….

Yet runs only on Windows XP

What has to move?

There are lots of apps that will only run on Windows XP



Old stuff has to move!

- Anything written in VB 6.0 or earlier
- Apps that use .NET 1.0, 1.1 and 2.0
- Older versions of Powerbuilder
- Older versions of Borland Delphi
- Apps using obsolete controls and components
- Apps with DLL hell problems.

There are three main legacy technologies to think about

There are lots of gotchas but they fall in three buckets:

- VB6 applications (millions of these!)
- Apps that use .NET 1.1
- Apps that only run on 32-bit processors

Each of these types of apps requires different solutions.

VB6 components need to be moved

Based on real-world VB6 to .NET migrations, here's what you need to fix:

Third party components were the most popular feature of legacy VB apps. You either have to replace with new (preferably 64-bit) components, new code (where no component is available) or access through COM Interop.

VB6 syntax changes

VB is loosely typed and all .NET languages (like C# and VB.NET) are strongly typed. The variant type in VB doesn't work in C#, and some operations on variants will not make sense in a strongly typed world. Replace those variants with explicit types and casts.

VB6 forms are different

The VB forms package is not WinForms. .NET uses either WinForms or WPF (Windows Presentation Foundation). If you're used to VB Forms, the WinForms designer will seem familiar, but it's still different.

.NET has issues but is backward compatible

You have a couple of things to worry about when upgrading from .NET 1.1 to current versions.

Breaking changes: although backwards compatible, all .NET versions "break" some behavior of prior releases. Your code may compile but not work correctly. Full testing is critical, as is understanding what parts of the framework are different. Here are three areas you should investigate:

- Breaking Changes in .NET Framework 2.0
- Changes in .NET Framework 3.5 SP1
- NET Framework 4 Migration Issues

You'll get .NET errors

All .NET 1.1 code should compile for .NET 4.5.1, but you will get compiler warnings on everything that's been declared obsolete or deprecated. You should examine each of these to decide if you need to make changes or if the old code is still acceptable.

32bit is hard to work around

Windows XP apps are running on old, limited, 32-bit hardware. You could upgrade it to make it suitable for Windows 8. But the cost would be more than replacing the PC altogether: a new CPU probably requires a new motherboard, more memory, bigger power supply, and a bigger hard drive.

Running XP in Windows 7 compatibility mode doesn't get you off the compliance hook, nor does it guarantee security. This is also true running inside any other VM like the Windows hypervisor or VMWare Workstation.

How can you get your apps off XP?

Many ways to move your apps off XP.

- But most solutions are:
  - Too expensive
  - Too slow for the business
  - Too technologically compromised
- Let's review the options:
  - Manual rewrite: Replicate existing functionality for modern platforms
  - Do nothing: Keep running the legacy application
  - Packaged software: Purchase commercial off-the-shelf LOB apps
  - Automation tools: Automated code conversion to new platform

The problem with manual rewrites

- High failure rate
  - 70% of manual software rewrites fail.*
- High cost
  - Costs 4 times more than migration.
- Huge defect rate
  - New code = 20 to 50 bugs per KLOCs.
  - 1,000,000 lines of new code: 20-50K bugs.
- Feature creep kills manual rewrites
  - Projects overcome by too many features.

*Standish Group, 2010

Other solutions have issues

Do nothing:

- Security and compliance issues like Sarbanes-Oxley, HIPAA and PCI
- High cost in maintenance, talent and resources

Packaged software:

- Business forced to adapt to applications
- Cost is known, but implementation cost is unpredictable
- Re-training of employees and IT to support

Why migrate old source code?

1. Automation creates no new bugs
2. Preserves business value.
3. New platform for future enhancement.
4. Refactored to support web architecture (MVVM).
5. Easier to maintain.
6. Easier to find trained developers.
7. Cheaper and faster than a rewrite.

What should you do?

You have alternatives to manual rewrites.

You can reuse existing functionality without starting over from scratch. Because you don't have to re-invent the wheel: Costs are lower. You need less time. Your risk is lower. No new bugs are introduced and no re-training is required because UI is the same (or similar).

And don’t forget about web architecture

The application can be re-factored and re-architected via automation tools to make the new application multi-tier and cloud-enabled.

It gets better

Once the code conversion is complete, you can enhance the app with new features, updated UI and other improvements.

You get a full-functioning application that runs on the new platform.

Do you want to know what it takes to move to a modern platform? Run our assessment tool and find out:

http://mobilize.hs-sites.com/html5-code-analyzer

Need more help?

Let a Mobilize.Net migration engineer help you figure out how to convert your legacy application: http://mobilize.net/talk-to-an-engineer/