

# Visual C++ 6.0 – Article!!!!!!

**Source:** <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.mfc/2004-08/0583.html>

---

**From:** Raj Sharma ([rakesh.bohara\\_at\\_gmail.com](mailto:rakesh.bohara_at_gmail.com))

**Date:** 08/10/04

Date: 10 Aug 2004 03:36:56 -0700

"Computers in the future may weigh no more than 1.5 tons."

Popular Mechanics, 1949

## Visual C++ 6.0

This is a brief review of Visual C++6.0, which was stuck on a 'scratch' machine to see what the installation effects would be before a full roll out was risked. Summary: a pretty easy upgrade, no real warnings and with some definite benefits.

1/Oct/98, May 99 Revised after more extended use.

### Installation

For some reason the installer does each install step which requires a reboot individually, rather than together. So it took three reboots before the real installation started.

It was pretty seamless after that. Caveats:

Don't uninstall source safe, as the Enterprise Edition installer updates it instead. Still, the installation was a lot better than the Studio 97 process.

Use the devkey utility to save your keys, and maybe even save the studio registry tree yourself.

Copy any macros, customised usertype.dat and stuff out of the old directories before letting the Studio97 uninstaller loose.

Don't let source safe 6.0 upgrade a shared database until everyone using it has upgraded.

### IDE

The IDE looks pretty much the same as usual. This is great –it minimises learning time. But if the IDE hasn't changed, why, why, why did the key bindings and settings not get transferred? I had to waste half an hour getting my key bindings back the way I liked them.

The new command line completion is cool, but takes some getting used to. The tooltips showing function arguments is particularly good –it

makes up for the on line help being even worse than before.

### C/C++ Compiler

There are no major changes here. What is nice is the new warning messages on common typos –the extra semicolon after an if: and the use of == in an assignment. For example, the following broken code snippet would raise two warnings:–

```
if (test);  
    a==b;
```

The optimiser does not seem significantly different. What is wierd is a new \_\_assume() keyword, where you can tell the optimiser certain facts hold. This is a nice idea, which could be expanded on in future. Imagine telling the compiler that in a function which took two pointer arguments, they would never point to the same address? Or that string pointers passed in were DWORD aligned? Those are the kind of things advanced optimisers would love to know. But for now \_assume() seems pretty limited, and there are few hints as to what it likes to know.

The assembler finally handles P5 functions like "cpuid" and P6/PII opcodes, specifically the CMOV conditional move series. This is great for hand tuned, maximum performance routines. Of course, things would be easier if the compiler could generate CMOV instead of branches on simple assignments, or did "modern" optimisation techniques –specifically loop unrolling. But no: for those you have to get the VTUNE compiler off Intel.

Performance wise the compiler seems no slower than before; maybe even a bit faster. I hope the editor doesn't slow down on a 'low end' Pentium with all the command line completion. Even on a fast box the class window flickers all the time in a way which soon gets really irritating. This is probably a sign of someone using a white background brush when a NULL brush would have been more appropriate.

Testing the optimiser on a large, performance critical app was not successful –it kept on crashing with some error saying a pure virtual function was being called. Since the app is out in the field, there seems no point upgrading to 6.0 just to go through the entire QA process again, so no time has been invested to chase down the problem.

Out in the field there are rumours that the new MFC42.DLL breaks old apps, and the MS knowledge base documents how a more brittle C run time library –MSVCRT.DLL– breaks some shipping product, including the Office 97 SP1 upgrade of OLE2's RPC sub system (RPCRSS) on Win98 boxes with DUN. When installing Visual Studio you agree to imdenify MS against any damage redistributing their libraries cause. So not only to you take the support call when a user installing your DLL based app ends up with a broken PC, you can end up at the wrong end of a lawsuit. Fix: build with the static libraries.

One thing not tested is how well the compiler can be used for building and debugging WDM device drivers. The NT5 DDK says 'yes, but we haven't tested it fully'.

#### Debugger

This product's killer feature for app development is not the debugger itself, but the 'edit and continue' function. After editing code the compiler can rebuild the application and update the memory image. This is a major productivity boost, especially for implementing user interface functionality, where a bit of trial and error is an inevitable part of the prototyping effort. It still isn't as good as an interpreted system: compilation takes time after all, but it does reduce the round trip time of the edit–compile–debug cycle.

#### Wizards

There are now app wizards for basic DLLs and applications. Not profound, but useful nonetheless. They are very time saving when knocking up short WinMain() based applications, as they eliminate almost all the drudge work before the coding begins. And the MFC app wizard gives you the choice of Doc/View or not. This is profound –many of the MFC utility classes have till now assumed that there was a CDocument derived class somewhere, and that AfxGetCurrentDocument() kept on getting called in places like CToolBar. For document free MFC apps to work, a lot of reengineering must have taken place. This is great when you are using it to write an OCX.

#### MFC

As mentioned above, MFC has a document free mode. Handy. And it has all the IE4 preview classes too. But why, why, why doesn't it come with a CDib class for decent image support? Borland wrote one of these for OWL 2.0 way back in 1994, and now, four years later MFC, the dominant C++ class library, still hasn't come up with an equivalent. This is simply ridiculous.

#### Header Files

The Windows header files are now much more interesting. Define WINVER as 0x500 and up pops API calls for Win98 and NT5 –albeit pre beta two header files. Seeing what happens on an NT5 beta 2 upgrade will be an interesting experiment.

The new 'getting ready for Win64' header file is there, if you so want to use it. No sign of COM+; I suspect when that comes out you'll need a new compiler and linker too –or some serious preprocessor. Presumably these will be drop in parts, with wizard support.

#### OCX Library

This hasn't been fully explored yet, but it looks like there are some new OCX wrappers around many of the IE4 common controls, all in a new DLL–MSCTL2.DLL. Things like the date/time picker should be good for use in Access and Excel 'apps', as well as VB.

### ATL3.0

ATL has evolved significantly, and the Wizards with it. It is now probably as easy as it will ever be to write COM interfaces and classes in C++ –without a major rework of the C++ language. That rework is scheduled for COM+, which leaves a bit of a question mark over ATL. If you want to do COM work today then it's great, but there's going to be some major code maintenance come COM+.

### Help Files

It was probably Programmer's Work Bench in Microsoft C6.0 that was their first 'IDE' that integrated documentation with the editor, so that F1 would explain a function. Since then there have been ups and downs. The early Windows based IDEs (Visual C++ 1.x– 2.x) all showed the help files in a pop up window, with a fair bit of navigation needed to flip between them. Then version 4 came out, which integrated help with the IDE. Visual C++ 5 continued this great tradition, replacing the help file format with HTML to slow everything down and cause no end of grief whenever beta copies of Explorer get installed. This was perceived by most developers as a retrograde step. Studio 6.0 takes a giant leap backwards by reverting to a separate window for the documentation, and it's the dire new HTML help UI, with toolbar buttons the size of small windows and no way to customise the keyboard for easy mouse-free navigation. This is a sad day, and a classic example of a technology driven rather than customer driven product development process. Remember: it is better to learn from other's mistakes, than to repeat them.

If there is one silver lining, the new command line completion helps you avoid the new help system until you really need to.

### Tools

I was disappointed to find that Source Safe 6.0's 'enhanced mobility support' means that the on line documents resort to telling you how to edit a file you haven't checked out by flipping the RO bit, then hacking it back in to the system on your return (the hard part). Anyone who's developed on a laptop will have learned about this a long time ago. I was really expecting something that worked better over dial up links by implementing a proper client/server system and not implementing the database as a shared file system –an approach that went out of style a long time ago. Still, there is one advantage to not having your source maintained by an MTS/COM based server with a Web interface: you know it'll be relatively safe. And for developing on the move, a mobile phone is a great accessory –not so much for data access as telling other engineers which files to check out on your behalf, and which to leave alone.

Visual Modeller 2.0 looks kind of interesting, and ties in well with my recent project post mortem conclusions of "be more rigorous during design". So it's learn UML time. Although a cut down version of Rational Software's Rose product, the full blown ISV tool won't integrate with VB6 until the Rose 98i upgrade comes out some time. The

Enterprise Edition tool pops up with a 3 tier model, and can incorporate the whole MFC class library to simplify 1 to 3 tier coding.

All the usual SDK tools get bundled in as expected, and work pretty well.

#### Documentation

Printed documentation is almost entirely absent. The Enterprise Edition upgrade comes with a thin booklet on Enterprise application development, which covers three-tier models and various product development lifecycles, before delving into the details of database access. This second half of the book tries to explain when to use which database access paradigm: ODBC, DAO, RDO or ADO/OLE DB.

There are already books in the retail channel which cover VC6: an update to Kruglinski's Inside Visual C++ and various others. Some of these may be worthwhile even if you are upgrading from previous editions: there have been a lot of subtle changes going on to MFC.

#### Complaints

Various complaints, although the way the class view flickers all the time is really, really irritating. And the help system well, it's not a good advert for HTML is it? We still remember the VC4.x documentation system—the best on line help a windows compiler ever had. NB, the help system is 'better' when you copy the entire MSDN disk set to a hard disk –something that is worthwhile doing if it can be shared by a whole development team.

Yet another database access API is getting predictable and boring. Given that the 'blessed' paradigm gets updated on an eighteen month basis, the only sensible option is actually to split all DB access into a separate class/module/thread for easy future maintainence. Once you've done that you can worry about which paradigm to use. OLE DB looks the best long term option, as it seems the only one to come out recently which was actually designed, rather than documented after implementation. But it is probably best to avoid until it killer bugs have been found by other developers, rather than yourself.

#### Service Pack 1

A lot of PCs broke when VC6 came out, mainly because the run time libraries were less forgiving to badly behaved shipping applications –including parts of the OS. It's incidents like that that make you want to build statically linked applications, regardless of what it does to footprints.

The service pack fixes these problems, so with it you can now use VC6 for developing production code. The optimiser is still buggy in 'maximum speed' mode. Our moderately sized 40KLOC app with major timing constraints was cajoled into building this mode by locating the one routine which was causing the problem and turning off all

optimisation there. The troublesome code doesn't look to suspicious: three boolean variables are used to choose which of 5 objects to create and use as the implementation of an interface, but the compiler seems to overwrite the register containing the pointer just before it gets invoked. After this bug was worked around, we got 3–9% speedup in some time critical code. This was good, but not enough to merit requalifying the entire application on its own, which the errant optimiser effectively mandates. Nine percent is the about the same speed up an Intel Xeon processor has over a PII at the same clock speed, so to many users the optimisation boost could be justified –but as usual you probably need to add a few extra features to justify upgrade dollars.

#### Analysis

Technical review over, now it's time to set the strategy and ruthless bits and look at the product with a more critical and cynical eye. Please skip this section if you don't want one developer's opinions.

#### Strategic Analysis

Visual Studio 6.0 will be the last IDE roll before NT5 and DNA. It has to become the tool to ensure not only that Windows remains the only platform of clients, but to help ensure that NT Server can replace the mainframe. Does it?

Sort of.

The C++ compiler doesn't make it much easier to write COM apps than the previous version. ATL and the Wizards help somewhat, but it's going to be COM3/COM+ where the fun begins. And that is not yet ready for prime time. What the tools do do is make it trivial to deliberately or accidentally write code which only runs on an IE4 enhanced box. Well, a ubiquitous web browser component does make life easier for developers, but not all users may take the same view, especially considering NT3.51 still crops up server side on a regular basis. Given that IE is winning the battle for browser share, and many apps (Outlook 98, Quicken 98) upgrade systems whether they want it or not, within 12 months programs will probably be able to ship just assuming that IE is there. Till then, life is complicated. Ship the 30+ MB installation on your CD and you end up fielding the support calls. Omit it and you get a different set of calls, especially from users with little or no Internet access who are not prepared to wait or pay for a download. The easiest solution is going to be to bundle the common controls upgrade and avoid the IE4 web control.

Next is the question as to whether it will enhance Microsoft's market share and revenue in the Development tools market. Well, apart from the Java product category, what else is happening in this market? Delphi has a following –but it's size and growth are unclear, and although C++ Builder is a good UI development tool, it seems not to have enough of a customer base to guarantee it a future. So, in the absence of the any major competition, the Windows app market is pretty

much owned by Microsoft. This may be a good thing –it guarantees consistency between tools– but I recall the effect that decent competition from Borland had on Microsoft –it forced them to evolve from a DOS based development system to Visual C++ within two years. C++ builder clearly isn't forcing a similar evolution this time round –it's in the Java space that the R&D dollars have been thrown.

The Java IDE is a blatant attack on the cross platform nature of Java, but this product's success depends on the outcome of all the legal battles and the adoption of the non portable language and classes. The vocal part of the Java community –naive little optimists that they are at times– are certainly against Visual J++. But the target of Visual J++ is clearly not those folk, but the enterprise developers –the authors of the mission critical in house applications which let employees fill in expense forms, bank tellers to query the accounts of clients, or airlines to issue tickets. Whoever owns those apps, owns the desktop. Microsoft are certainly offering those developers a path –it will be up to them to choose whether to follow it.

#### Cost/Benefit Analysis

The cost of a full upgrade is in the 600–700 (dollars/pounds/euros) category. For that you get a new set of tools which takes time to install (1 hour) and more time to get to grips with –assume another day to learn what the new features in the IDE are, plus a week per class library or language. Then there is the indeterminate amount of time to QA existing code –and start chasing down any issues the optimiser raises. And finally, you may decide this is the time to get a new PC/HDD or RAM boost. Over all the initial cost of the new toolkit is going to be in the \$1500–\$2K mark per head, mostly due to lost time. That investment has to be recouped by the team producing better code, where better can be measured along axes such as performance, quality, features or TTM. Performance and quality are questionable in an x.0 release of any compiler (Borland's track record here make the MS tools seem a bastion of stability). Once the class libraries and optimisers have settled down improvements here start to arise. Till then the only gain is Time To Market. Well, the C++ IDE is certainly better here than before. Command line completion is brilliant –it's taken its time to move over from VB, and it's a great day now that it has. In the absence of any quantitative data, I'll take a guess at a 10% coding speed up with this. Thinking time is the same, but helpfile and class heirarchy searching is down. The other performance boost is edit–and–continue. This is a significant speedup, as it reduces the cycle time of the edit–compile–run process, a cycle time which grows with the size of an app. So, if you feel that edit and continue will help you, then go for it. And for small to medium programs it probably will. Large apps may benefit from it the most, but here a little red flag pops up.

Any program of significant size –50+KLOC, for example– has usually been built by a number of people and is too complex for a single developer to start editing at random. That's why change control

processes usually get instituted, along with formal defect tracking systems. Edit and continue does not seem entirely compatible with such processes. In fact, its sole purpose is practically to give the over-subscribed "code and fix" development lifecycle greater performance and hence legitimacy. So, while it's nice to have when hacking away on recreational and exploratory apps, the benefit of that feature really depends on what your development processes are –the more informal the better, by the sign of things.

The other fly in the ointment of the cost/benefit analysis is that the full studio product costs the same as it did when Studio 97 came out. Since that last product was introduced, the PC hardware business has been evolving rapidly. "Segment 0" –the \$999 PC– did not exist as far as Intel was concerned, and the under-specced Compaq box with the Cyrix GX superintegrated core an object of much curiosity. Now, eighteen months later \$999 will get you a decent 300 MHz core with hard disk, memory and 3D acceleration to match, and even \$799 will get you something reasonable like a Celeron 300. Any of the latest low cost PCs are up to spec as far as development goes, especially if you can haggle the DRAM up over 64MB in exchange for getting a system without the OS preloaded. (with MSDN Pro you get Windows anyway, so can 'opt' not to accept the preloaded OS's license terms for the refund). So whereas last year Visual Studio was half the price of the PC needed to run it, now the hardware and software costs are at parity. Doesn't seem such value for money in those terms, does it?

To summarise: before paying for a full studio upgrade, ask yourself which would deliver a bigger productivity boost: function completion or a second PC? You could work at home for less stress at weekends, or bring up beta operating systems on a box you don't depend on. Or spend slightly less time looking through the help files. Tough choice isn't it?

Extended Use Update: May '99

Six months worth of full time use, and you can really start to assess the value and robustness of the product. Now the toolkit has been used for the whole gamut of apps, from NT/WDM device drivers to SQL7 database analysis and the code behind web pages, with a bit of VB in between. The fact that you can use the Visual Studio to do the entire soup of tasks expected from a modern software engineer is testament to its breadth and flexibility – and it shows just how complex life for us developers have become.

I still get the impression I'm scratching the surface of what the system can do: I haven't invested any effort in studio plug ins, new wizards and stuff like that, or really got into what VB6 offers. Ole DB is kind of interesting, once you understand it. A single unified API/toolkit for scrolling through recordsets –from database queries to appointments in a diary– should make it a learn–once–use–everywhere programming skill. Ultimately, it will be profound: the Web Folders feature in IE5 uses it to read and write web site hosted documents, so it can even replace standard file IO as we know and love it. It's just

a pity OLE DB is so badly documented, with only a few terse examples, and its equally unfortunate that you can waste half an hour writing error querying code only to discover that Access' generic error message is 'An error occurred'.

Stability wise, the VC side it reasonably bulletproof, especially on NT. VJ++/Visual Interdev is too brittle to be usable, and can crash while doing basic editing. This is unacceptable for a 'professional' product. VB is something in between, in my experience. One app I have will cause VB to hang on the second debug run, so it's exit and restart the IDE every go –the use of the DHTML editing OCX may be the factor here. Performance wise, the tools are 'OK' for client side stuff, and an Intel VTune back end (\$300+) really boosts PIII code generation. Web side, you have to question ASP performance and scalability: for serious work its VC in an IIS plug in talking to a remote SQL server. There has to be a better way, but I don't think it's Java or Perl either.