Memory Bank

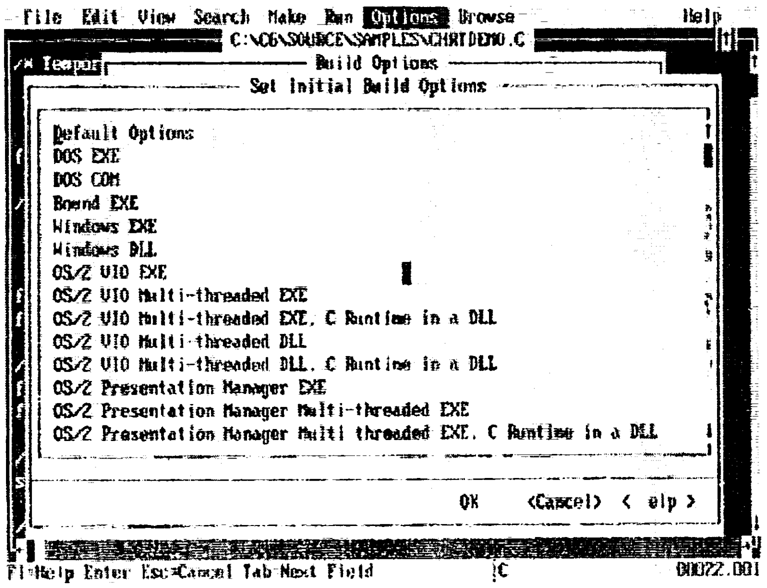
By Nancy Michell

Microsoft Programmer's WorkBench, 1990

Software products that were groundbreaking when they were introduced undoubtedly seem primitive 10 years later. That phenomenon doesn't really exist in other industries. Think about it. The car you drive today isn't 20 times faster than the one you drove in 1990, and the garage you keep it in isn't 80 times smaller. But in the software industry change happens at the speed of light. It's amazing to look back at products introduced only 10 years ago. They almost always look like they're straight out of a 1950s sci-fi flick.

Visual Studio® .NET, which you'll be hearing a lot about in this issue, is a direct descendant of Microsoft® Programmer's WorkBench (PWB). PWB was introduced in 1990, and like other software products of that vintage, it looks rather ancient by today's standards. Programmer's WorkBench was a character-based integrated development environment (IDE), so its "interface" is a bit frightening to all of us who have forgotten the days before graphics beckoned to us to point and click around the PC.

But looks can be deceiving. PWB was actually pretty advanced—it combined an editor, debugger, compiler, linker, a header file utility, and other features all in one "application." Having all these features in one place was a time-saver and a productivity booster, and just simply made the process more pleasant. For example, PWB included a feature that searched for files on a disk. While most developers had utilities they could run to accomplish this task, PWB allowed them to do it without leaving the environment—a real time-saver. Another handy feature was error message browsing. After the build, you could step through the build messages and fix each one as you went along. This really made it easy to locate bugs. PWB also included the first version of Microsoft C that supported OS/2, the operating system that was a joint-venture project between Microsoft and IBM.



```
C:\C6\SOURCE\SAMPLES\CHRTDEMO.C
Build Options
Set Initial Build Options

Default Options
DOS EXE
DOS COM
Bound EXE
Windows EXE
Windows DLL
OS/2 VIO EXE
OS/2 VIO Multi-threaded EXE
OS/2 VIO Multi-threaded EXE, C Runtime in a DLL
OS/2 VIO Multi-threaded DLL
OS/2 VIO Multi-threaded DLL, C Runtime in a DLL
OS/2 Presentation Manager EXE
OS/2 Presentation Manager Multi-threaded EXE
OS/2 Presentation Manager Multi threaded EXE, C Runtime in a DLL

F1=Help Enter Esc=Cancel Tab=Next Field          C          00022.001
```

Default Build Options in PWB

In a March 1990 article in *Microsoft Systems Journal* entitled "Microsoft C Version 6.0 Provides an Integrated Development," Noel J. Bergman describes PWB like this:

> If you can imagine the Microsoft Editor enhanced both internally and with extensions to resemble the Microsoft Quick C Version 2.0 environment, you have a rough idea of what PWB is like to work with.

John Norwood, who is currently lead program manager for Visual Studio .NET, reached back into his own memory bank to recall some of his favorite features in PWB. Two of the things he really liked were that it "had a very configurable build system and a lot of features."

> If you go back before PWB you get to the M editor. You could build, step through compile errors, and all sorts of stuff in the M editor, which was really the first IDE from Microsoft. Both M and PWB could be considered "super-editor" proto-IDEs but you'll find a lot of these outside of Microsoft. Emacs for example. (How far back does that go?)

PWB was considered slow by some, even at the time. But then so much was dependent on your system configuration. One developer commented back then that PWB was painfully slow unless "you had a fast 386 machine." Apparently his 16MHz 286 setup just didn't cut it.

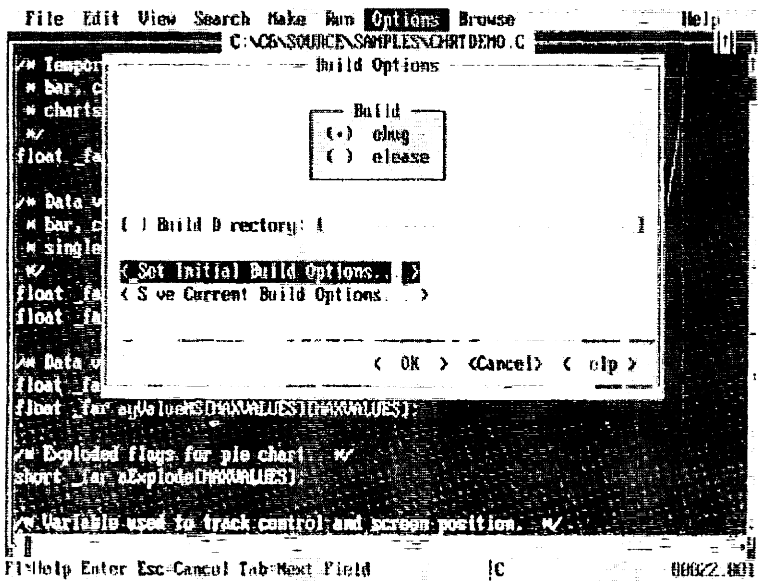
System Requirements for Microsoft C Version 6.0

Taking a look at the following system requirements for running Programmer's WorkBench, you'll see that if you did have a 386 you were ahead of the pack. Anybody remember 5¼ floppy disks? In case you've forgotten, they held 360KB. Well, once upon a time movies cost a nickel and you got free pottery. Times change.

- MS-DOS or PC-DOS 3.0 or higher or OS/2 version 1.1 or higher
- 640KB memory (1MB recommended) for MS-DOS
- 3MB memory (4MB recommended) for OS/2
- 34KB extended memory recommended for debugging large MS-DOS applications
- 8MHz 80286-based processor
- One high-density 5¼ or one 3½ double-sided disk drive
- 5MB hard drive
- 10MB disk space recommended-depends on memory models and libraries selected

Although much of the software of the time was slow, the M editor wasn't, and this caused problems for some developers who were upgrading to PWB. According to John:

> The original PWB was considered big, fat, and slow by M editor users. A lot of people refused to make the switch because they wouldn't take the speed hit. M was almost instantaneous in loading even on 1 meg 386s or 286s. The M editor (MEP: the 32-bit POSIX version) is still better than any (including current) Microsoft editors in loading large files because it has its own memory virtualization system.
> But the dev team tuned PWB, and the next version was really almost as fast as M. The real problem was load time because PWB had to parse a really complex configuration file.


File Edit View Search Make Run Options Browse Help
C:\C6\SOURCE\SAMPLES\CHRTDEMO.C
Build Options
Build
(•) ebug
( ) elease
[ ] Build D rectory: [ ]
< Set Initial Build Options... >
< S ve Current Build Options... >
< OK > <Cancel> < elp >
F1=Help Enter Esc=Cancel Tab=Next Field

```
/* Tempo
 * bar, c
 * charts
 */
float _fa

/* Data v
 * bar, c
 * single
 */
float _fa
float _fa

/* Data v
float _fa
float _far ayValue[MAXVALUES][MAXVALUES];

/* Exploded flags for pie chart. */
short _far aExplode[MAXVALUES];

/* Variable used to track control and screen position. */
```

Setting Build Options

Eventually, John said, it was delay loaded, and that made a huge difference in speed. It must certainly have helped, because Shankar Vaidyanathan, SDE Lead at Microsoft, remembers a speedy PWB:

> One thing that PWB was doing was seeking and loading up the buffer on a need basis, and so we used to be able to open large files very efficiently. I remember Notepad, Edit, and various text editors including that of Visual C++® 1.0 read the file all at once. So I remember firing up the PWB for large files on Win16 machines and it used to be blazingly fast.

If you're wondering if PWB was the first IDE from Microsoft, the answer is, well, sort of, according to John. But either way it was chock full of innovations, a few of which would be welcomed today.

> If you define an IDE as a combination of editor and debugger then PWB was the first for Microsoft, but only in a very loose sense since it really just spawned CodeView® as a separate process. The real thing that pushed the technology of the time was the CodeView debugger—it was fantastic. I think there may

> have been features that don't even exist in the current Visual Studio debugger, such as history. It would record every operation in debugging and you could play it back (even rewind the entire debug state) to get to a hard-to-reproduce bug. Horribly slow, but worth it when you had to have it. The memory window was also an elegant implementation. I don't know if people in Microsoft realize how crappy the debugging experience is on other platforms and how much our state-of-the-art debuggers have contributed to the success of our platforms, and to the success of people who develop for our platforms.

All great eras must come to an end. Microsoft C 7.0 was the last version that shipped with PWB, according to Shankar. But for developers it was a long-awaited version:

> C 7.0 was so delayed that our customers were waiting with bated breath for a long while to get their hands on this. Also C/C++ 7.0 was the first time we had C++ compiler going.

If you look at this first IDE from Microsoft, you can envision the evolutionary process that has led up to the current version of Visual Studio .NET. Take a look at "Talking To Rob Brigham" on page 6 for clues on how this happened. Happy hunting!

Nancy Michell is associate editor at MSDN Magazine where she spends her time manipulating bits of string literals for the good of mankind.

From the July/August 2001 issue of MSDN News.

Back to top