On the large number of ways of expressing Microsoft Visual C++ compiler versions

devblogs.microsoft.com/oldnewthing/20221219-00

December 19, 2022

Raymond Chen

When you ask someone what version of the Microsoft Visual C++ compiler they're using, or if somebody tells you "This feature requires version X of the Microsoft Visual C++ compiler", you can get the answer is a large number of ways, because the Microsoft Visual C++ compiler has apparently decided that if one version number is good, then more must be better.

The first version number that enters the picture is the name of the Visual Studio product the compiler comes with. This is probably something like "Visual Studio YYYY" for some year, like "Visual Studio 2019".

The next version number is the product version of that Visual Studio product. For example, "Visual Studio 2019 version 16.11."

The next version number is the platform toolset that you specify in your project file, like `<PlatformToolset>v142</PlatformToolset>`.

Another version number is the actual toolchain version, like "14.29".

And then there's the version number reported by the `_MSC_VER` predefined macro, like "1929".

On top of that is the version number reported by the `_MSC_FULL_VER` predefined macro, like "192930100".

And finally, there's the version number reported by the compiler itself when you type `cl /?`.

How are all of these version numbers related?

Product name	⇔	Product version	⇔	Platform toolset	⇔	Toolchain	⇔	`_MSC_VER`	⇔	`_MSC_FULL_VER`	⇔	Co bar

Okay, let's do the easy one first: The last three version numbers are related to each other in a straightforward way. If the compiler's self-reported version is `aa.bb.ccccc.dd`, then the `_MSC_VER` is `aabb` and the `_MSC_FULL_VER` is `aabbccccc`. In other words, `_MSC_VER = aa * 100 + bb` and `_MSC_FULL_VER = aa * 10000000 + bb * 100000 + cc`.

Okay, so that lets us build a relationship among the last three boxes: The compiler banner is the basis for the other two.

The first two boxes are also related, but in a less obvious way: The product name and product major version line up according to this table on Wikipedia. For example, Visual Studio 2019 corresponds to product versions 16.*.

The next two boxes also appear to be related, although I can't find any official documentation to that effect. The platform toolset appears to be the letter "v", followed by the toolchain major version, followed by the first digit of the minor version. For example, a toolchain version of "14.29" corresponds to a platform toolset of "v142".

That leaves three major categories: The Visual Studio product, the toolchain, and the compiler. These three categories follow their own path, so you have to use a cheat sheet to see how they correspond to each other.[1]

Let's build that cheat sheet. I got the raw data from this table on Wikipedia, with gaps filled in from the archived Visual Studio release notes.

Name	Product	Toolset	Toolchain	`_MSC_VER`	`_MSC_FULL_VER`	Compiler
Visual Studio 2017	15.0	v141	14.1	1910	1910xxxxx	19.10.xxxxx
	15.1					
	15.2					
	15.3		14.11	1911	1911xxxxx	19.11.xxxxx
	15.4					
	15.5		14.12	1912	1912xxxxx	19.12.xxxxx
	15.6		14.13	1913	1913xxxxx	19.13.xxxxx
	15.7		14.14	1914	1914xxxxx	19.14.xxxxx
	15.8		14.15	1915	1915xxxxx	19.15.xxxxx
	15.9		14.16	1916	1916xxxxx	19.16.xxxxx

Visual Studio 2019	16.0	v142	14.20	1920	1920xxxxx	19.20.xxxxx
	16.1		14.21	1921	1921xxxxx	19.21.xxxxx
	16.2		14.22	1922	1922xxxxx	19.22.xxxxx
	16.3		14.23	1923	1923xxxxx	19.23.xxxxx
	16.4		14.24	1924	1924xxxxx	19.24.xxxxx
	16.5		14.25	1925	1925xxxxx	19.25.xxxxx
	16.6		14.26	1926	1926xxxxx	19.26.xxxxx
	16.7		14.27	1927	1927xxxxx	19.27.xxxxx
	16.8		14.28	1928	1928xxxxx	19.28.xxxxx
	16.9					
	16.10		14.29	1929	1929xxxxx	19.29.xxxxx
	16.11					
Visual Studio 2022	17.0	v143	14.30	1930	1930xxxxx	19.30.xxxxx
	17.1		14.31	1931	1931xxxxx	19.31.xxxxx
	17.2		14.32	1932	1932xxxxx	19.32.xxxxx
	17.3		14.33	1933	1933xxxxx	19.33.xxxxx
	17.4		14.34	1934	1934xxxxx	19.34.xxxxx

But wait, the story isn't over yet.

Internally, the compiler team delivers periodic compiler updates to the Windows team. These updates are named LKG followed by a number, like "LKG14", and there is an internal Web site that maps LKG values to compiler version numbers. Fortunately, only people who work at Microsoft need to worry about these LKG version numbers.

Bonus chatter: The term LKG stands for "Last Known Good", meaning that it is the latest version of the compiler that has been validated against the Windows code base. There is another term FKG, which you think might stand for "First Known Good", but it doesn't. It originally stood for "Fast Known Good" because it contained compilers even newer than the Last Known Good. That policy has changed, and now the FKG is used for other purposes, but the name FKG stuck, even though it's completely wrong.

[1] It appears that starting in Visual Studio 2017, the compiler minor version increases by one each time the toolchain minor version increases by one, so that's handy. Starting in August 2017, the compiler version is equal to the toolchain version plus five. I don't know whether this is a rule or a coincidence.

Raymond Chen

Follow