What do the numbers in a version typically represent (i.e. v1.9.0.1)?

Asked 16 years, 3 months ago Modified 3 months ago

Viewed 196k times

202

I've always assumed each number delineated by a period represented a single component of the software. If that's true, do they ever represent something different?

How should a version number be structured to start assigning versions to the different `builds` of my software? As an aside, my software has five distinct components.

version

Share

Improve this question

Follow

edited Dec 6, 2022 at 15:12

ΩmegaMan

31.4k ● 12 ● 107 ● 133

asked Sep 15, 2008 at 19:01

BeachRunnerFred

18.5k ● 35 ● 143 ● 240

1 Numbers can mean anything you want, although they are usually not related to individual components but rather to major vs. minor vs. maintenance changes in your release.

Check out these resources: netbeans.org/community/guidelines/process.html en.wikipedia.org/wiki/Release_engineering freebsd.org/releases/6.0R/schedule.html Cheers – Alvaro Rodriguez Sep 15, 2008 at 19:04

You mean e.g. not i.e. in the title – Howard Apr 10, 2023 at 20:05

I have never encountered software which used a version number where every number delineated by a period represents a single component of the software and I have never expected a version number to be represented that way. It's a complete surprise to find someone who does. – NeilG Jul 7, 2023 at 0:31

27 Answers

Sorted by: Highest score (default)

295

In version *1.9.0.1*:

- **1**: Major revision (new UI, lots of new features, conceptual change, etc.)
- **9**: Minor revision (maybe a change to a search box, 1 feature added, collection of bug fixes)
- **0**: Bug fix release
- **1**: Build number (if used)—that's why you see the .NET framework using something like 2.0.4.2709

You won't find a lot of apps going down to four levels, 3 is usually sufficient.

Share Improve this answer edited Aug 29, 2013 at 7:20

Follow

Constantino Tsarouhas
6,886 ● 6 ● 44 ● 54

answered Sep 15, 2008 at 19:04

Dillie-O
29.7k ● 14 ● 102 ● 141

3 I use exactly this, but specifically the Build number is the Subversion Database repository version – Xetius Sep 15, 2008 at 20:24

1 I use the same, but without the third digit, as in major.minor.build. The reason being the build number will increase anyway, so that in itself can identify the fact minor bugfixes etc have taken place. – Mark Embling Jun 4, 2009 at 22:54

11 major.minor.revision(bug fixes).build makes the most sense to me. Unfortunately, the .NET Version type is defined as major.minor.build.revision (possibly because Microsoft used to only use 3 version places?). – Kevin Kibler Nov 14, 2009 at 3:47

2 I'm trying to understand this system. So here is a question: What if a new release has a feature and a bug fix, what should I increment? – iTurki Jul 16, 2013 at 19:36

9 @iturki Typically the "larger" version number takes precedence. So if you're updating your app from version 1.4.23, you simply update to 1.5.0 and be done with it. You can indicate in your release notes which bugs have been fixed. Similarly, you can update from 1.4.23 to 2.0.0. – Dillie-O Jul 16, 2013 at 21:39

There is the Semantic Versioning specification

70

This is the summary of version 2.0.0:

> Given a version number MAJOR.MINOR.PATCH, increment the:
>
> 1. MAJOR version when you make incompatible API changes,
> 2. MINOR version when you add functionality in a backwards-compatible manner, and
> 3. PATCH version when you make backwards-compatible bug fixes.
>
> Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.

Share Improve this answer

Follow

edited Mar 3, 2020 at 4:20

Benjamin W.

51.6k ● 19 ● 125 ● 131

answered Jul 24, 2014 at 18:38

magyk

701 ● 5 ● 2

20

It can be very arbitrary, and differs from product to product. For example, with the Ubuntu distribution, 8.04 refers to 2008.April

Typically the left most (major) numbers indicate a major release, and the further you go to the right, the smaller

the change involved.

Share Improve this answer

Follow

answered Sep 15, 2008 at 19:03

rkabir

568 ● 4 ● 14

13

major.minor[.maintenance[.build]]

http://en.wikipedia.org/wiki/Software_versioning#Numeric

Share Improve this answer

Follow

answered Sep 15, 2008 at 19:04

Søren Spelling Lund

1,652 ● 1 ● 13 ● 18

11

Numbers can be useful as described by other answers, but consider how they can also be rather meaningless... Sun, you know SUN, java: 1.2, 1.3, 1.4 1.5 or 5 then 6. In the good old Apple II version numbers Meant Something. Nowadays, people are giving up on version numbers and going with silly names like "Feisty fig" (or something like that) and "hardy heron" and "europa" and "ganymede". Of course this is far less useful because, you're going to run out of moons of jupiter before you stop changing the program, and since there's no obvious ordering you can't tell which is newer.

Share Improve this answer

Follow

answered Sep 15, 2008 at 20:05

stu

8

The more points, the more minor the release. There's no real solid standard beyond that - can mean different things based on what the project maintainers decide on.

WordPress, for example, goes along these lines:

1.6 -> 2.0 -> 2.0.1 -> 2.0.2 -> 2.1 -> 2.1.1 -> 2.2 ...

1.6 to 2.0 would be a big release - features, interface changes, major changes to the APIs, breakage of some 1.6 templates and plugins, etc. 2.0 to 2.0.1 would be a minor release - perhaps fixing a security bug. 2.0.2 to 2.1 would be a significant release - new features, generally.

Share Improve this answer

Follow

answered Sep 15, 2008 at 19:05

ceejayoz

180k ● 41 ● 308 ● 380

5

Usually its:

MajorVersion.MinorVersion.Revision.Build

Share Improve this answer

Follow

answered Sep 15, 2008 at 19:04

Restore The Data Dumps Again

39.3k ● 12 ● 99 ● 124

4

Version numbers don't usually represent separate components. For some people/software the numbers are fairly arbitrary. For others, different parts of the version number string do represent different things. For example, some systems increase parts of the version number when a file format changes. So V 1.2.1 is file format compatible with all other V 1.2 versions (1.2.2, 1.2.3, etc.) but not with V 1.3. Ultimately it's up to you what scheme you want to use.

Share Improve this answer

Follow

answered Sep 15, 2008 at 19:05

user9385

41 ● 2

4

release.major.minor.revision would be my guess.
But it can vary greatly between products.

Share Improve this answer

Follow

edited Nov 16, 2016 at 8:37

Bondax

3,163 ● 1 ● 29 ● 35

answered Sep 15, 2008 at 19:03

Fire Lancer

30.1k ● 33 ● 122 ● 184

2

It depends, but the typical representation is that of *major.minor.release.build*.

Where:

- *major* is the major release version of your software, think .NET 3.x
- *minor* is the minor release version of your software, think .NET x.5
- *release* is the release of that version, typically bugfixes will increment this
- *build* is a number that denotes the number of builds you have performed.

So for instance, 1.9.0.1, means that it's version 1.9 of your software, following 1.8 and 1.7, etc. where 1.7, 1.8 and 1.9 all in some way typically add small amounts of new features alongside bugfixes. Since it's x.x.0.x, it's the initial release of 1.9, and it's the first build of that version.

You can also find good information on the Wikipedia article on the subject.

Share Improve this answer

Follow

answered Sep 15, 2008 at 19:06

Lasse V. Karlsen

391k ● 106 ● 646 ● 844

2

Major.Minor.Bugs

(Or some variation on that)

Bugs is usually bug fixes with no new functionality.

Minor is some change that adds new functionality but doesn't change the program in any major way.

Major is a change in the program that either breaks old functionality or is so big that it somehow changes how users should use the program.

Share Improve this answer

Follow

answered Sep 15, 2008 at 19:06

emeryc

825 ● 7 ● 8

2

Everyone chooses what they want to do with these numbers. I've been tempted to call releases a.b.c since it's rather silly anyway. That being said, what I've seen over the last 25+ years of development tends to work this way. Let's say your version number is 1.2.3.

The "1" indicates a "major" revision. Usually this is an initial release, a large feature set change or a rewrite of significant portions of the code. Once the feature set is determined and at least partially implemented you go to the next number.

The "2" indicates a release within a series. Often we use this position to get caught up on features that didn't make it in the last major release. This position (2) almost always indicates a feature add, usually with bug fixes.

The "3" in most shops indicates a patch release/bug fix. Almost never, at least on the commercial side, does this indicate a significant feature add. If features show up in position 3 then it's probably because someone checked something in before we knew we had to do a bug fix release.

Beyond the "3" position? I have no clue why people do that sort of thing, it just gets more confusing.

Notably some of the OSS out there throws all this out of wack. For example, Trac version 10 is actually 0.10.X.X. I think a lot of folks in the OSS world either lack confidence or just don't want to announce that they have a major release done.

Share Improve this answer Follow

answered Sep 15, 2008 at 19:10

mclain

2

The paradigm of major release.minor release.bug fix is pretty common, I think.

In some enterprise support contracts there is $$$ (or breach of contract liability) associated with how a particular release is designated. A contract, for example, might entitle a customer to some number of major releases in a period of time, or promise that there will be fewer than x number of minor releases in a period, or that support will continue to be available for so many releases. Of course no matter how many words are put in to the contract to explain what a major release is versus a minor release, it is always subjective and there will always be gray areas – leading to the possibility that the software vendor can game the system to beat such contractual provisions.

Share Improve this answer

Follow

answered Sep 15, 2008 at 19:59

Will M

2,491 ● 2 ● 18 ● 19

2

Despite the fact that most of the previous answers give perfectly good explanations for how version numbering could and should be used, there is also another scheme, which I would call the *marketing versioning scheme*. I'll add this as an answer, because it exists, not because I think it's worth following.

In the *marketing versioning scheme*, all those technical thoughts and meanings are replaced by **bigger is better**. The version number of a product is derived from two rules:

- bigger (higher) numbers sell better than smaller (lower) numbers
- our version number should be bigger (higher) than any of the competitors' version numbers

That takes version numbering out of the hands of the technical staff and puts where it belongs (sales and marketing). [irony warning]

However, since a technical version still makes sense in a way, the marketing versions are often accompanied under the hood by technical version numbers. They are usually somehow hidden, but can be revealed by some *info* or *about* dialog.

Share Improve this answer Follow

edited Jul 7, 2023 at 6:47

answered Feb 22, 2022 at 17:17

not2savvy

4,153 ● 4 ● 33 ● 53

There are also secondary rules involved, such as "thousands are better than hundreds" which can mean that a version number jumps a large increment suddenly. Another one is "more zeroes is better than fewer zeroes". And also "repetition is better". There's a subtle interplay between these rules, often dictated by competitor's numbering. So for instance version numbers could have reached `101` but then suddenly jump to `1001`, then possibly a small increment to `1010` then straight to `2000` or even `2010` or `2020`. – NeilG Jul 7, 2023 at 0:40

1

Major.minor.point.build usually. Major and minor are self-explanatory, point is a release for a few minor bugfixes, and build is just a build identifier.

Share Improve this answer Follow

answered Sep 15, 2008 at 19:04

Serafina Brocious

30.6k ● 12 ● 91 ● 115

What is a build identifier? – Darshan L Aug 27, 2018 at 7:40

A combination of major, minor, patch, build, security patch, etc.

The first two are major & minor-- the rest will depend on the project, company and sometimes community. In OS's like FreeBSD, you will have 1.9.0.1_number to represent a security patch.

Share Improve this answer

Follow

answered Sep 15, 2008 at 19:04

Loren Segal

3,272 ● 1 ● 30 ● 29

Yup. Major releases add big, new features, may break compatibility or have significantly different dependencies, etc.

Minor releases also add features, but they're smaller, sometimes stripped-down ported versions from beta major release.

If there is a third version number component, it's usually for important bugfixes, and security fixes. If there are more, it really depends so much on product that it's difficult to give general answer.

Share Improve this answer

Follow

answered Sep 15, 2008 at 19:05

Paweł Hajdan

18.5k ● 9 ● 52 ● 65

1

Generally then number are in the format of version.major.minor.hotfix, not individual internal components. So v1.9.0.1 would be version 1, major release 9 (of v1), minor release (of v1.9) 0, hot fix 1 of (v1.9.0).

Share Improve this answer

Follow

answered Sep 15, 2008 at 19:07

Scott Bevington

1,201 ● 10 ● 12

1

From the C# AssemblyInfo.cs file you can see the following:

```
// Version information for an assembly consists of
the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
/ You can specify all the values or you can
default the Build and Revision Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("1.0.*")]
```

Share Improve this answer

Follow

answered Sep 15, 2008 at 19:11

Thomas Jespersen

11.8k ● 15 ● 50 ● 59

People don't always recognize the subtle difference between version numbers like 2.1, 2.0.1, or 2.10 - ask a

1

technical support person how many times they've had trouble with this. Developers are detail oriented and familiar with hierarchical structures, so this is a blind spot for us.

If at all possible, expose a simpler version number to your customers.

Share Improve this answer

Follow

answered Sep 25, 2008 at 17:20

Mark Ransom

308k ● 44 ● 416 ● 647

1

In the case of a library, the version number tells you about the **level of compatibility** between two releases, and thus how difficult an upgrade will be.

A bug fix release needs to preserve binary, source, and serialization compatibility.

Minor releases mean different things to different projects, but usually they don't need to preserve source compatibility.

Major version numbers can break all three forms.

I wrote more about the rationale here.

Share Improve this answer

Follow

answered Nov 28, 2010 at 16:34

Craig P. Motlin

26.7k ● 18 ● 103 ● 127

0

Depends a bit on the language, Delphi and C# for example have different meanings.

Usually, the first two numbers respresent a major and a minor version, i.e. 1.0 for the first real release, 1.1 for some important bugfixes and minor new features, 2.0 for a big new feature release.

The third number can refer to a "really minor" version, or revision. 1.0.1 is just a very small bugfix to 1.0.0 for example. But it can also carry the Revision number from your Source Control System, or an ever-incrementing number that increments with every build. Or a Datestamp.

A little bit more detail here. "officially", in .net, the 4 numbers are "Major.Minor.Build.Revision", whereas in Delphi there are "Major.Minor.Release.Build". I use "Major.Minor.ReallyMinor.SubversionRev" for my versioning.

Share Improve this answer

Follow

edited May 23, 2017 at 12:18

Community Bot

1 • 1

answered Sep 15, 2008 at 19:06

Michael Stum

181k • 119 • 407 • 541

The first number is typically referred to as the major version number. It's basically used to denote significant changes between builds (i.e. when you add many new

0

features, you increment the major version). Components with differing major versions from the same product probably aren't compatible.

The next number is the minor version number. It can represent some new features, or a number of bug fixes or small architecture changes. Components from the same product which differ by the minor version number may or may not work together and probably shouldn't.

The next is usually called the build number. This may be incremented daily, or with each "released" build, or with each build at all. There may be only small differences between two components who differ by only the build number and typically can work well together.

The final number is usually the revision number. Often times this is used by an automatic build process, or when you're making "one-off" throw-away builds for testing.

When you increment your version numbers is up to you, but they should always *increment* or *stay the same*. You can have all components share the same version number, or only increment the version number on changed components.

Share Improve this answer

Follow

answered Sep 15, 2008 at 19:09

Bob King

25.8k ● 7 ● 57 ● 66

0

The version number of a complex piece of software represents the whole package and is independent of the version numbers of the parts. The Gizmo version 3.2.5 might contain Foo version 1.2.0 and Bar version 9.5.4.

When creating version numbers, use them as follows:

1. First number is main release. If you make significant changes to the user interface or need to break existing interfaces (so that your users will have to change their interface code), you should go to new main version.
2. Second number should indicate that new features have been added or something works differently internally. (For example the Oracle database might decide to use a different strategy for retrieving data, making most things faster and some things slower.) Existing interfaces should continue working and the user interface should be recognizable.
3. Version numbering further is up to the person writing the software - Oracle uses five (!) groups, ie. an Oracle version is something like 10.1.3.0.5. From third group down, you should only introduce bugfixes or minor changes in functionality.

Share Improve this answer

Follow

answered Sep 15, 2008 at 19:11

Sten Vesterli

3,033 ● 1 ● 23 ● 22

0

the ones that vary less would be the first two, for major.minor, after that it can be anything from build, revision, release, to any custom algorithms (like in some MS products)

Share Improve this answer

Follow

answered Sep 15, 2008 at 19:14

BlackTigerX

6,136 ● 7 ● 41 ● 50

0

Every organization/group has it's own standard. The important thing is that you stick to whatever notation you choose otherwise your customers will be confused. Having said that I've used normally 3 numbers:

x.yz.bbbbb. Where: x: is the major version (major new features) y: is the minor version number (small new features, small improvements without UI changes) z: is the service pack (basically the same as x.y but with some bug fixes bbbb: is the build number and only really visible from the "about box" with other details for customer support. bbbb is free format and every product can use it's own.

Share Improve this answer

Follow

answered Sep 15, 2008 at 19:26

Vasco Duarte

279 ● 5 ● 12

It is sometimes the goal to confuse the customer into buying the new version. See this answer:

stackoverflow.com/a/71225569/134044 – NeilG Jul 7, 2023 at 0:42

0

Here is what we use:

1. First number = Overall system era. Changes every couple of years and typically represents a fundamental change in technology, or client features or both.
2. Second number = database schema revision. An increment in this number requires a database migration and so is a significant change (or systems replicate and so changing the database structure requires a careful upgrade process). Resets to 0 if the first number changes.
3. Third number = software only change. This can usually be implemented on a client by client basis as the database schema is unchanged. Resets to zero if the second number changes.
4. Subversion version number. We populate this automatically on build using the TortoiseSVN tool. This number never resets but continually increments. Using this we can always recreate any version.

This system is serving us well because every number has a clear and important function. I have seen other teams grappling with the major number/minor number question (how big a change is major) and I dont see the benefit to

that. If you dont need to track database revisions just go to a 3 or 2 digit version number, and make life easier!

Share Improve this answer

Follow

answered Sep 15, 2008 at 19:48

Ewan Makepeace

5,416 ● 9 ● 33 ● 31