

[Scala 3 Reference](#) / [Language Versions](#) / [Binary Compatibility](#)

LEARN

INSTALL

PLAYGROUND

FIND A LIBRARY

COMMUNITY

BLOG

# Binary Compatibility


[Edit this page on GitHub](#)

In Scala 2 different minor versions of the compiler were free to change the way how they encode different language features in JVM bytecode so each bump of the compiler's minor version resulted in breaking binary compatibility and if a project had any Scala dependencies they all needed to be (cross-)compiled to the same minor Scala version that was used in that project itself. On the contrary, Scala 3 has a stable encoding into JVM bytecode.

In addition to classfiles the compilation process in Scala 3 also produces files with `.tasty` extension. The **TASTy** format is an intermediate representation of Scala code containing full information about sources together with information provided by the typer. Some of this information is lost during generation of bytecode so Scala 3 compilers read TASTy files during compilation in addition to classfiles to know the exact types of values, methods, etc. in already compiled classes (although compilation from TASTy files only is also possible). TASTy files are also typically distributed together with classfiles in published artifacts.

TASTy format is extensible but it preserves backward compatibility and the evolution happens between minor releases of the language. This means a Scala compiler in version `3.x1.y1` is able to read TASTy files produced by another compiler in version `3.x2.y2` if  $x1 \geq x2$  (assuming two stable versions of the compiler are considered - **SNAPSHOT** or **NIGHTLY** compiler versions can read TASTy in an older stable format but their TASTy versions are not compatible between each other even if the compilers have the same minor version; also compilers in stable versions cannot read TASTy generated by an unstable version).

TASTy version number has the format of `<major_version>.<minor_version>-<experimental_version>` and the numbering changes in parallel to language releases in such a way that a bump in language minor version corresponds to a bump in TASTy minor version (e.g. for Scala `3.0.0` the TASTy version is `28.0-0`). Experimental

version set to 0 signifies a stable version while others are considered unstable/experimental. TASTy version is not strictly bound to the data format itself - 

any changes to the API of the standard library also require a change in TASTy minor version.

Being able to bump the compiler version in a project without having to wait for all of its dependencies to do the same is already a big leap forward when compared to Scala 2. However, we might still try to do better, especially from the perspective of authors of libraries. If you maintain a library and you would like it to be usable as a dependency for all Scala 3 projects, you would have to always emit TASTy in a version that would be readable by everyone, which would normally mean getting stuck at 3.0.x forever.

To solve this problem a new experimental compiler flag `-scala-output-version <version>` (available since 3.1.2) has been added. Setting this flag makes the compiler produce TASTy files that should be possible to use by all Scala 3 compilers in version `<version>` or newer. This flag was inspired by how `-java-output-version` (formerly `-release`) works for specifying the target version of JDK. More specifically this enforces emitting TASTy files in an older format ensuring that:

- the code contains no references to parts of the standard library which were added to the API after `<version>` and would crash at runtime when a program is executed with the older version of the standard library on the classpath
- no dependency found on the classpath during compilation (except for the standard library itself) contains TASTy files produced by a compiler newer than `<version>` (otherwise they could potentially leak such disallowed references to the standard library).

If any of the checks above is not fulfilled or for any other reason older TASTy cannot be emitted (e.g. the code uses some new language features which cannot be expressed in the older format) the entire compilation fails (with errors reported for each of such issues).

As this feature is experimental it does not have any special support in build tools yet (at least not in sbt 1.6.1 or lower). E.g. when a project gets compiled with Scala compiler `3.x1.y1` and `-scala-output-version 3.x2` option and then published using sbt then the standard library in version `3.x1.y1` gets added to the project's dependencies instead of `3.x2.y2`. When the dependencies are added to the classpath during compilation with Scala `3.x2.y2` the compiler will crash while trying to read TASTy files in the newer format. A currently known workaround is to modify

the build definition of the dependent project by explicitly overriding the version of Scala standard library in dependencies, e.g.



```
dependencyOverrides += Seq(  
  scalaOrganization.value %% "scala3-library" % scalaVersion.value,  
  scalaOrganization.value %% "scala3-library_sjs1" % scalaVersion.value //  
  for Scala.js projects  
)
```

The behaviour of `-scala-output-version` flag might still change in the future, especially it's not guaranteed that every new version of the compiler would be able to generate TASTy in all older formats going back to the one produced by `3.0.x` compiler.

[← Source ...](#)

[Soft Ke... >](#)