



Scala 3 Reference

[LEARN](#)[INSTALL](#)[PLAYGROUND](#)[FIND A LIBRARY](#)[COMMUNITY](#)[BLOG](#)

Scala 3 Reference

[✎ Edit this page on GitHub](#)

Scala 3 implements many language changes and improvements over Scala 2. In this reference, we discuss design decisions and present important differences compared to Scala 2.

Goals

The language redesign was guided by three main goals:


- Strengthen Scala's foundations. Make the full programming language compatible with the foundational work on the [DOT calculus](#) and apply the lessons learned from that work.
- Make Scala easier and safer to use. Tame powerful constructs such as implicits to provide a gentler learning curve. Remove warts and puzzlers.
- Further improve the consistency and expressiveness of Scala's language constructs.

Corresponding to these goals, the language changes fall into seven categories: (1) Core constructs to strengthen foundations, (2) simplifications and (3) [restrictions](#), to make the language easier and safer to use, (4) [dropped constructs](#) to make the language smaller and more regular, (5) [changed constructs](#) to remove warts, and increase consistency and usability, (6) [new constructs](#) to fill gaps and increase expressiveness, (7) a new, principled approach to metaprogramming that replaces [Scala 2 experimental macros](#).

Essential Foundations

These new constructs directly model core features of DOT, higher-kinded types, and the [SI calculus for implicit resolution](#).

- [Intersection types](#), replacing compound types,
- [Union types](#)

- [Union types](#),
- [Type lambdas](#), replacing encodings using structural types and type projection. 
- [Context functions](#), offering abstraction over given parameters.

Simplifications

These constructs replace existing constructs with the aim of making the language safer and simpler to use, and to promote uniformity in code style.

- [Trait parameters](#) replace [early initializers](#) with a more generally useful construct.
- [Given instances](#) replace implicit objects and defs, focussing on intent over mechanism.
- [Using clauses](#) replace implicit parameters, avoiding their ambiguities.
- [Extension methods](#) replace implicit classes with a clearer and simpler mechanism.
- [Opaque type aliases](#) replace most uses of value classes while guaranteeing absence of boxing.
- [Top-level definitions](#) replace package objects, dropping syntactic boilerplate.
- [Export clauses](#) provide a simple and general way to express aggregation, which can replace the previous facade pattern of package objects inheriting from classes.
- [Vararg splices](#) now use the form `xs*` in function arguments and patterns instead of `xs: _*` and `xs @ _*`,
- [Universal apply methods](#) allow using simple function call syntax instead of `new` expressions. `new` expressions stay around as a fallback for the cases where creator applications cannot be used.

With the exception of [early initializers](#) and old-style vararg patterns, all superseded constructs continue to be available in Scala 3.0. The plan is to deprecate and phase them out later.

Value classes (superseded by opaque type aliases) are a special case. There are currently no deprecation plans for value classes, since we might bring them back in a more general form if they are supported natively by the JVM as is planned by [project Valhalla](#).

Restrictions

These constructs are restricted to make the language safer.

- [Implicit Conversions](#): there is only one way to define implicit conversions instead of many, and potentially surprising implicit conversions require a

instead of many, and potentially surprising implicit conversions require a language import.



- **Given Imports:** implicits now require a special form of import, to make the import clearly visible.
- **Type Projection:** only classes can be used as prefix `C` of a type projection `C#A`. Type projection on abstract types is no longer supported since it is unsound.
- **Multiversal Equality:** implement an "opt-in" scheme to rule out nonsensical comparisons with `=` and `≠`.
- **infix:** make method application syntax uniform across code bases.

Unrestricted implicit conversions continue to be available in Scala 3.0, but will be deprecated and removed later. Unrestricted versions of the other constructs in the list above are available only under `-source 3.0-migration`.

Dropped Constructs

These constructs are proposed to be dropped without a new construct replacing them. The motivation for dropping these constructs is to simplify the language and its implementation.

- **DelayedInit,**
- **Existential types,**
- **Procedure syntax,**
- **Class shadowing,**
- **XML literals,**
- **Symbol literals,**
- **Auto application,**
- **Weak conformance,**
- Compound types (replaced by **Intersection types**),
- **Auto tupling** (implemented, but not merged).

The date when these constructs are dropped varies. The current status is:

- Not implemented at all:
 - DelayedInit, existential types, weak conformance.
- Supported under `-source 3.0-migration`:
 - procedure syntax, class shadowing, symbol literals, auto application, auto tupling in a restricted form.
- Supported in 3.0, to be deprecated and phased out later:
 - **XML literals**, compound types.

Changes



These constructs have undergone changes to make them more regular and useful.

- **Structural Types**: They now allow pluggable implementations, which greatly increases their usefulness. Some usage patterns are restricted compared to the status quo.
- **Name-based pattern matching**: The existing undocumented Scala 2 implementation has been codified in a slightly simplified form.
- **Automatic Eta expansion**: Eta expansion is now performed universally also in the absence of an expected type. The postfix `_` operator is thus made redundant. It will be deprecated and dropped after Scala 3.0.
- **Implicit Resolution**: The implicit resolution rules have been cleaned up to make them more useful and less surprising. Implicit scope is restricted to no longer include package prefixes.

Most aspects of old-style implicit resolution are still available under `-source 3.0-migration`. The other changes in this list are applied unconditionally.

New Constructs

These are additions to the language that make it more powerful or pleasant to use.

- **Enums** provide concise syntax for enumerations and **algebraic data types**.
- **Parameter untupling** avoids having to use `case` for tupled parameter destructuring.
- **Dependent function types** generalize dependent methods to dependent function values and types.
- **Polymorphic function types** generalize polymorphic methods to polymorphic function values and types. *Current status*: There is a proposal and a merged prototype implementation, but the implementation has not been finalized (it is notably missing type inference support).
- **Kind polymorphism** allows the definition of operators working equally on types and type constructors.
- **@targetName annotations** make it easier to interoperate with code written in other languages and give more flexibility for avoiding name clashes.

Metaprogramming

The following constructs together aim to put metaprogramming in Scala on a new basis. So far, metaprogramming was achieved by a combination of macros and

basis. So far, metaprogramming was achieved by a combination of macros and libraries such as [Shapeless](#) that were in turn based on some key macros. Current Scala

2 macro mechanisms are a thin veneer on top the current Scala 2 compiler, which makes them fragile and in many cases impossible to port to Scala 3.

It's worth noting that macros were never included in the [Scala 2 language specification](#) and were so far made available only under an `-experimental` flag. This has not prevented their widespread usage.

To enable porting most uses of macros, we are experimenting with the advanced language constructs listed below. These designs are more provisional than the rest of the proposed language constructs for Scala 3.0. There might still be some changes until the final release. Stabilizing the feature set needed for metaprogramming is our first priority.

- [Match Types](#) allow computation on types.
- [Inline](#) provides by itself a straightforward implementation of some simple macros and is at the same time an essential building block for the implementation of complex macros.
- [Quotes and Splices](#) provide a principled way to express macros and staging with a unified set of abstractions.
- [Type class derivation](#) provides an in-language implementation of the `Gen` macro in Shapeless and other foundational libraries. The new implementation is more robust, efficient and easier to use than the macro.
- [By-name context parameters](#) provide a more robust in-language implementation of the `Lazy` macro in [Shapeless](#).

See Also

[A classification of proposed language features](#) is an expanded version of this page that adds the status (i.e. relative importance to be a part of Scala 3, and relative urgency when to decide this) and expected migration cost of each language construct.

New Ty... >

Contributors to this page



pikinier20



BarkingBad



julienrf



michelou



rjolly



ShapelessCat



AugustNagro



b-studios



LPTK



odersky



asakaev



robstoll



rhumbertgz



Scaladoc

Copyright (c) 2002-2022, LAMP/EPFL

