Go 1.27 Release Notes

- Introduction to Go 1.27 {#introduction}
- Changes to the language {#language}
- Tools {#tools}
  - Go command {#go-command}
    - GODEBUG
    - go test
    - go doc
    - go fix
    - go mod tidy
  - Gofmt {#gofmt}
  - Vet {#vet}
  - Trace
- Runtime {#runtime}
  - Faster memory allocation
  - Goroutine leak profile {#goroutineleak-profiles}
- Compiler {#compiler}
- Linker {#linker}
- Standard library {#library}
  - New encoding/json/v2 and encoding/json/jsontext packages {#jsonv2}
  - New crypto/mldsa package {#crypto_mldsa}
  - New uuid package {#uuid}
  - New experimental simd package {#simd}
  - Experimental simd/archsimd package {#archsimd}
  - Minor changes to the library {#minor_library_changes}
    - bytes
    - compress/flate
    - crypto
    - crypto/ecdsa

    - crypto/tls
    - crypto/x509
    - crypto/x509/pkix
    - database/sql
    - database/sql/driver
    - go/constant
    - go/scanner
    - go/token
    - go/types
    - hash/maphash
    - math/big
    - math/rand/v2 {#rand}
    - net
    - net/http
    - net/http/httptest
    - net/url
    - runtime/secret
    - strings
    - syscall
    - testing/synctest
    - unicode
- Ports {#ports}
    - Darwin {#darwin}
    - PowerPC {#ppc64}

Introduction to Go 1.27 {#introduction}

The latest Go release, version 1.27, arrives in August 2026, six months after Go 1.26. Most of its changes are in the implementation of the toolchain, runtime, and libraries. As always, the release maintains the Go 1 promise of compatibility. We expect almost all Go programs to continue to

compile and run as before.

Changes to the language {#language}

Go 1.27 now supports generic methods: a method declaration may declare its own type parameters. This widely anticipated change allows adding generic functions within the namespace of a particular data type where before one had to declare such functions with a scope of the entire package. As an example, `math/rand/v2` now also declares a generic method with signature `(*Rand) N[Int intType](Int) Int`, whereas previously it had just a generic function `N[Int intType](Int) Int` (see below for details). Note that methods of interfaces may not declare type parameters nor can interface methods be implemented by generic methods.

A key in a struct literal may now be any valid field selector for the struct type, not just a (top-level) field name of the struct.

Function type inference has been generalized to apply in all contexts where a generic function is assigned to a variable of (or converted to) a matching function type.

Tools {#tools}

Response file (`@file`) parsing is now supported for the `compile`, `link`, `asm`, `cgo`, `cover`, and `pack` tools. The response file contains whitespace-separated arguments with support for single-quoted and double-quoted strings, escape sequences, and backslash-newline line continuation. The format is compatible with GCC's response file implementation to ensure interoperability with existing build systems.

Go command {#go-command}

The `go` command no longer has support for the `bzr` version control system. It will no longer be able to directly fetch modules hosted on `bzr` servers.

`GODEBUG`

Starting with Go 1.27, the `go` command now recognizes a `GODEBUG` setting for which support was removed (such as `asynctimerchan`, see below) if it appears in `go.mod` files (`godebug` entries) and `.go`

source files (`//go:debug` comments). It accepts these settings if they are set to the final default value established before the setting was removed. If they are set to an old value, the `go` command will fail. This change is in the spirit of the Go 1 compatibility guarantee and allows existing programs that set supported `GODEBUG` settings to continue to build and run without changes even when the respective setting support has been removed.

`go test`

`go test` now invokes the `stdversion` vet check by default. This reports the use of standard library symbols that are too new for the Go version in force in the referring file, as determined by `go` directive in `go.mod` and build tags on the file.

`go test -json` now annotates `"Action":"output"` lines with an optional new field `"OutputType"`, specifying the type of output. Currently, the possible values include "error", "error-continue", and "frame". See cmd/test2json help for details.

`go doc`

The `go doc` command now supports `package@version` syntax, such as `go doc example.com/pkg@v1.2.3`.

The `go doc` command now accepts the `-ex` command-line option to list executable examples of the given package or symbol. When an example name is passed on the command line (such as `go doc bytes.ExampleBuffer`), `go doc` now prints the example source code along with comments.

`go fix`

The `go fix` command contains several new modernizers (`atomictypes`, `embedlit`, `slicesbackward`, and `unsafefuncs`).

The existing `fmtappendf` analyzer was removed due to stylistic concerns.

The existing `waitgroup` analyzer was renamed to `waitgroupgo` to avoid ambiguity.

`go mod tidy`

For modules specifying `go 1.27` or later in their `go.mod` file, `go mod tidy` now automatically merges duplicate require blocks. This ensures the file maintains a clean, standard structure containing at most two require blocks: one for direct dependencies and one for indirect dependencies.

Existing comment blocks attached to dependencies are preserved during this consolidation. If a comment block is associated with a mixed set of directives (containing both direct and indirect dependencies), the comment block is merged and attached to the new direct dependency block.

Previously, if a `go.mod` file accumulated multiple disjoint require blocks (often due to manual edits, unresolved Git merge conflicts, or legacy upgrades) `go mod tidy` would leave the extra blocks

intact or inadvertently create new ones. The tool now strictly enforces the two-block layout, consolidating disparate requirements into their respective blocks and cleaning up the structure of the module file automatically.

Gofmt {#gofmt}

This release includes a fix to the logic used by `gofmt` (and `go/format`) in deciding column alignment. The previous logic caused the behavior of `gofmt` to vary slightly across CPU architectures. The new logic changes the alignment decision for a small number of files, so running `gofmt` from Go 1.27 on previously formatted code may produce minor whitespace changes.

Vet {#vet}

The printf analyzer now reports calls such as `fmt.Errorf("...: %w", p)` in which the `%w` operand `p` has type `*E`, where the type `E` itself implements `error`. Wrapping a pointer in this way is usually a latent bug since it causes spurious failures of operations that test for a particular error type, such as a type assertion or call to `errors.Is`.

Trace

`go tool trace`'s `-http` command-line option now restricts the listen address to localhost when passed only a port (e.g., `-http=:6060`). This change makes `go tool trace` consistent with the behavior of `go tool pprof`'s `-http` flag. To listen on all addresses, explicitly include the specified address (e.g., `-http=0.0.0.0:6060`).

Runtime {#runtime}

Tracebacks for modules with `go` directives configuring Go 1.27 or later will now include `runtime/pprof` goroutine labels in the header line. This behavior can be disabled with `tracebacklabels=0` `GODEBUG` setting (added in Go 1.26). This opt-out is expected to be kept indefinitely in case goroutine labels acquire sensitive information that shouldn't be made available in tracebacks.

The `asynctimerchan` `GODEBUG` setting (added in Go 1.23) has been removed permanently. Channels created by package time are now always unbuffered (synchronous), irrespective of `GODEBUG` settings.

Faster memory allocation

The compiler now generates calls to size-specialized memory allocation routines, reducing the cost of some small (<80 byte) memory allocations by up to 30%. Improvements vary depending on the workload, but the overall improvement is expected to be ~1% in real allocation-heavy programs.

This causes the binary size to increase by about 60 KB (independent of the workload). Please file an issue if you notice any regressions. You may set `GOEXPERIMENT=nosizespecializedmalloc` at build time to disable it. This opt-out setting is expected to be removed in Go 1.28.

Goroutine leak profile {#goroutineleak-profiles}

A new profile type that reports leaked goroutines, previously available as an experiment in Go 1.26, is now generally available. The new profile type, named `goroutineleak`, is supported in the `runtime/pprof` package. It is also available as the `net/http/pprof` endpoint `/debug/pprof/goroutineleak`.

A *leaked* goroutine is a goroutine blocked on some concurrency primitive (channels, `sync.Mutex`, `sync.Cond`, etc) that cannot possibly become unblocked. The runtime detects leaked goroutines using the garbage collector: if a goroutine G is blocked on concurrency primitive P, and P is unreachable from any runnable goroutine or any goroutine that *those* could unblock, then P cannot be unblocked, so goroutine G can never wake up. While it is impossible to detect permanently blocked goroutines in all cases, this approach detects a large class of such leaks.

Because this technique builds on reachability, the runtime may fail to identify leaks caused by blocking on concurrency primitives reachable through global variables or the local variables of runnable goroutines.

See Go 1.26 release notes for an example.

Special thanks to Vlad Saioc at Uber for contributing this work.

The `goroutineleakprofile` `GOEXPERIMENT` setting is now deleted.

Compiler {#compiler}

The compiler now resolves a relative filename in a `//line` or `/*line*/` directive against the directory of the file containing the directive, matching the behavior of `go/scanner`. Absolute filenames are unaffected. See #70478.

The compiler now generates simpler names for function literals (closures). Previously, when the containing function is inlined, the function literal's name can get quite long. Now the compiler chooses the same name for the function literal regardless of inlining. It may also combine multiple instances of the same function literal (as its containing function is inlined) to share the same code in the compiled binary. This change does not affect the functionality of Go code. Tests that check symbol names may need update, although it is recommended to not depend on the names of function literals. For programs that incorrectly compare function code pointer for equality, the issue may be more exposed with Go 1.27, as function literals with different captured closure data may have equal code pointers in more cases.

Linker {#linker}

When targeting macOS, the linker now accepts `-macos` and `-macsdk` command-line options, which specify the OS and SDK versions in the `LC_BUILD_VERSION` load command. By default, it selects the oldest supported macOS version (currently 13.0.0) and a recent SDK version (currently 26.2.0).

Standard library {#library}

New encoding/json/v2 and encoding/json/jsontext packages {#jsonv2}

Two new packages are now available:

- The `encoding/json/v2` package is a major revision of `encoding/json`. It provides `Marshal`, `MarshalWrite`, `MarshalEncode`, `Unmarshal`, `UnmarshalRead`, and `UnmarshalDecode`, all of which accept variadic `Options` arguments to configure marshaling and unmarshaling behavior.
- The `encoding/json/jsontext` package provides lower-level syntactic processing of JSON. The `Encoder` and `Decoder` types operate on JSON as a sequence of `Token` and `Value`, maintaining a state machine to ensure the produced or consumed sequence is valid JSON text.

The v2 package chooses stricter, more interoperable defaults than v1: it rejects invalid UTF-8 in JSON strings and rejects duplicate names within a JSON object. See the v1 `encoding/json` package documentation for the complete set of behavioral differences and the options available to adjust them.

The `encoding/json` package is now backed by the v2 implementation. Marshaling and unmarshaling behavior is preserved, but the exact text of error messages may differ. The package also gains a number of new `Options` that can configure v2 to operate with v1 semantics to avoid requiring a full migration to the new API. The v1 API will continue to be supported and users are not required to migrate.

Marshal performance is broadly at parity with the previous implementation, while unmarshal performance is significantly faster.

Users who encounter compatibility problems with the new implementation may disable it by setting `GOEXPERIMENT=nojsonv2` at build time, restoring the original v1 implementation. This opt-out is expected to be removed in a future release.

Some notable changes to `encoding/json/v2` over the course of the `GOEXPERIMENT` include the following removals: the `format` tag option (#79071), the `unknown` tag option (#77271), the `DiscardUnknownMembers` marshal option (#77271), and the `SkipFunc` sentinel error (#74324). In addition, the `inline` tag option was renamed to `embed` (#79985), and behavior was updated for the `string` tag option (#79065) and the `MatchCaseInsensitiveNames` option (CL 792780). In `encoding/json/jsontext`, the numeric `Token` accessors were changed to also return errors (#77666).

See the proposal issue for background and additional detail. If you need to disable the new implementation, please file an issue.

New crypto/mldsa package {#crypto_mldsa}

The new `crypto/mldsa` package implements the post-quantum ML-DSA signature scheme specified in FIPS 204.

`crypto/x509` now supports ML-DSA private keys, public keys, and signatures.

`crypto/tls` now supports ML-DSA signatures in TLS 1.3, with the new `MLDSA44`, `MLDSA65`, and `MLDSA87` `SignatureScheme` values.

New uuid package {#uuid}

The new `uuid` package generates and parses UUIDs.

New experimental simd package {#simd}

Go 1.27 introduces a new experimental `simd` package that provides portable and vector-size-agnostic SIMD support. It will make use of the hardware instructions if they are available. This package is enabled by setting the environment variable `GOEXPERIMENT=simd` at build time.

The `simd` package is available on all architectures, and provides vector types of unspecified size such as `Int8s` and `Float32s`. It supports a "scalable" subset of the operations present in the `simd/archsimd` package that are hardware-supported or easily emulated across architectures and vector widths.

See the proposal issue for more details.

Experimental simd/archsimd package {#archsimd}

Go 1.27 continues the experimental support for SIMD operations in the `simd/archsimd` package that began in Go 1.26. This release revises the amd64 API and adds support for arm64 "Neon" 128-bit SIMD and WebAssembly 128-bit SIMD. The `simd/archsimd` package is enabled by setting the environment variable `GOEXPERIMENT=simd` at build time.

This package provides access to architecture-specific SIMD operations. It supports 128-bit vector types on wasm, arm64, and amd64, and 256-bit and 512-bit vector types on some amd64 processors. The API is not yet considered stable.

See the package documentation and the proposal issue for more details.

We intend to provide support for additional architectures in future versions, but the API is intentionally architecture-specific and thus non-portable.

Minor changes to the library {#minor_library_changes}

`bytes`

The new `CutLast` function slices a `[]byte` around the last occurrence of a separator. It can replace and simplify some common uses of `LastIndex`.

compress/flate

Compression speed is improved in Go 1.27. The exact encoded output from `Writer` may be different from Go 1.26 as a result of the encoder implementation change. Since DEFLATE is the underlying compression used in `archive/zip`, `compress/gzip`, `compress/zlib`, and `image/png`, the outputs from those packages may also have changed.

crypto

The new `MLDSAMu` `Hash` value is added for use as a signaling mechanism for External μ ML-DSA signing.

crypto/ecdsa

`PrivateKey.Sign` now checks that the length of the hash is correct, if a non-nil `SignerOpts` is provided.

crypto/tls

The new `QUICConfig.ClientHelloInfoConn` field specifies the `net.Conn` to use for the `ClientHelloInfo.Conn` field during QUIC server handshakes.

The `MLKEM1024` key exchange is now supported. It can be enabled by adding it to `Config.CurvePreferences`.

`Config.Rand` is now deprecated. For deterministic testing, use `testing/cryptotest.SetGlobalRandom`.

Post-quantum hybrid key exchanges can now be explicitly enabled in `Config.CurvePreferences` even if the `tlsmlkem=0` or `tlssecpmlkem=0` `GODEBUG` options are used. Those options were always meant to only apply to the default set used when `Config.CurvePreferences` is nil.

The new `ConnectionState.LocalCertificate` field contains the certificate chain presented to the connection peer during the handshake.

The `tlsunsafeekm` (added in Go 1.22), `tlsrsakex` (added in Go 1.22), `tls3des` (added in Go 1.23), `tls10server` (added in Go 1.22), and `x509keypairleaf` (added in Go 1.23) `GODEBUG` settings have been removed permanently.

The `tlskyber` `GODEBUG` setting (added in Go 1.23) was removed in Go 1.24 but the removal was not documented in the release notes for Go 1.24 at that time; it is documented now.

crypto/x509

When parsing into `pkix.Name` fields, a wider range of `pkix.AttributeTypeAndValue.Value` types is now supported, and unknown types are parsed into `asn1.RawValue`.

The new `Certificate.RawSignatureAlgorithm`, `CertificateRequest.RawSignatureAlgorithm`, and `RevocationList.RawSignatureAlgorithm` fields expose the DER-encoded AlgorithmIdentifier of the signature algorithm, including when the SignatureAlgorithm field is `UnknownSignatureAlgorithm`.

`SystemCertPool` now respects SSL_CERT_FILE and SSL_CERT_DIR on Windows and Darwin. When these environment variables are set, roots are loaded from disk and instead of using the platform certificate verification APIs, the native Go verifier is used. This behavior can be disabled with `GODEBUG=x509sslcertoverrideplatform=0`.

crypto/x509/pkix

`RDNSequence.String` (and therefore `Name.String`) now renders string-typed attribute values as strings even when the attribute's OID is unrecognized. Previously such values were always hex-encoded in their DER form. See #33093.

database/sql

The new `ConvertAssign` function gives database drivers access to the type conversions performed by `Rows.Scan`.

database/sql/driver

Drivers may implement the new `RowsColumnScanner` interface to scan directly into user-provided destinations.

go/constant

The new `StringLen` function returns the length of a string `Value` without fully constructing the `Value`.

go/scanner

The scanner now allows retrieving the end position of a token via the new `Scanner.End` method.

go/token

`File` now has a `String` method.

go/types

The `Hasher` type is an implementation of `maphash.Hasher` for `Type`s that respects the `Identical` equivalence relation, allowing `Types` to be used in hash tables and similar data structures. `HasherIgnoreTags` is the analogous hasher for `IdenticalIgnoreTags`.

The `gotypesalias` `GODEBUG` setting (added in Go 1.22) has been removed permanently and the package `go/types` now always produces an `Alias` type node for alias declarations irrespective of `GODEBUG` settings.

hash/maphash

The `Hasher` interface type defines the contract between values of a particular type and future hash-based data structures such as hash tables and Bloom filters; see #70471.

The `ComparableHasher` type provides a convenient implementation of `Hasher` for comparable types where the `Equal` method is defined as `==`.

math/big

`Int` now has a `Divide` method to compute quotient and remainder of two `Int` values. It supports rounding modes `Trunc`, `Floor`, `Round`, and `Ceil`.

math/rand/v2 {#rand}

`Rand` now supports a generic method `N`, matching the behavior of the top-level `N` function.

net

`UnixConn` read methods now return `io.EOF` directly instead of wrapping it in `net.OpError` when the underlying read returns EOF.

net/http

`Transport` and `Server` support TLS ALPN protocol negotiation on user-provided `net.Conn` connections which implement a `ConnectionState() tls.ConnectionState` method.

HTTP/2 server now accepts client priority signals, as defined in RFC 9218, allowing it to prioritize serving HTTP/2 streams with higher priority. If the old behavior is preferred, where streams are served in a round-robin manner regardless of priority, `Server.DisableClientPriority` can be set to `true`.

HTTP/1 `Response.Body` now automatically drains any unread content upon being closed, up to a conservative limit, to allow better connection reuse. For most programs, this change should be a no-op, or result in a performance improvement. In rare cases, programs that do not benefit from connection reuse might experience performance degradation if they had been improperly allowing an excessive amount of idle connections to linger; usually by setting `Transport.MaxIdleConns` to `0` or using different `Client`s for different requests, thereby bypassing `Transport.MaxIdleConns` limit. In these cases, setting `Transport.DisableKeepAlives` to `true` will disable connection reuse. However, such performance degradation usually indicates improper configuration or usage of `Transport` or `Client` in the first place, and a deeper look would likely be beneficial.

The new `Server.MaxHeaderValueCount` field allows HTTP servers to control the number of header values that they are willing to accept. If unset, `DefaultMaxHeaderValueCount` is used.

net/http/httptest

The new `NewTestServer` function creates a `Server` configured to use an in-memory fake network suitable for use with the `testing/synctest` package.

net/url

The new `URL.Clone` method creates a deep copy of a URL. The new `Values.Clone` method creates a deep copy of Values.

runtime/secret

Goroutines that are created while in secret mode will now themselves execute in secret mode.

strings

The new `CutLast` function slices a string around the last occurrence of a separator. It can replace and simplify some common uses of `LastIndex`.

syscall

On Plan 9, the `Errno` type is now defined and implements the `error` interface, as on other platforms. Plan 9 system calls return `ErrorString` values, so `Errno` is never returned by this package on Plan 9. It is defined so that portable code referring to `syscall.Errno` builds on Plan 9 without build constraints.

testing/synctest

The new `Sleep` helper function combines `time.Sleep` and `synctest.Wait`.

unicode

The unicode package and associated support throughout the system have been upgraded from Unicode 15 to Unicode 17. See the Unicode 16.0.0 and Unicode 17.0.0 release notes for information about the changes.

Ports {#ports}

Darwin {#darwin}

As announced in the Go 1.26 release notes, Go 1.27 requires macOS 13 Ventura or later; support for previous versions has been discontinued.

PowerPC {#ppc64}

On the big-endian 64-bit PowerPC port on Linux (`GOOS=linux` `GOARCH=ppc64`), the Go toolchain now generates binaries that use the ELFv2 system ABI. ELFv2 support requires Linux kernel 3.13 or later. RHEL7 backported this support to its 3.10 kernel.

Cgo, position-independent executables (PIE), and external linking are now supported. Using these features requires an ELFv2 compatible runtime (libc and all linked and loaded libraries).

For programs that do not use cgo, the Go toolchain still generates static binaries with internal linking by default. For programs that have cgo options, if a static, pure-Go binary is needed, one can set the environment variable `CGO_ENABLED=0` when running `go build`.

□□□ #2
□ □□□ □□□ 2026-10-26 20:26:29 UTC
□ □□□ □□□ 2026-10-26 20:26:29 UTC