**The Way To V**

****

**By Nyx-litenite - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=83474099**

***A thorough introduction to the V programming language***

**Dr. Ivo Balbaert**

Copyright © Ivo Balbaert – 2019- .

# Preface

***Code less, compile quicker, execute faster => have more fun!***

How did V originate?

And so was born "V" or the V language, a language that has the feel of the Go language, but has a performance like C and a safety more like Rust.

This book is intended for developers who want to learn this fascinating and promising language.

Some basic knowledge of programming and some experience with a programming language and environment is assumed, but a thorough knowledge of Go or C is not needed.

For those of you who are familiar with C or the current object oriented languages, we will compare the concepts in V with the corresponding concepts in these languages (throughout the book we will use the well known “OO” abbreviation, to mean object-oriented).

This text explains everything from the basic concepts onwards, but at the same time we discuss advanced concepts and techniques.

[XXX To be adapted]

In Part I we discuss the origins of the language (Ch. 1) and get you started with the installation of Go (Ch. 2) and a development environment (Ch. 3).

Part 2 then guides you through the core concepts of Go: the simple and composite types (Ch. 4, 7, 8), control structures (Ch. 5), functions (Ch. 6), struts with their methods (Ch. 10), and interfaces (Ch. 11). The functional and object-oriented aspects of Go are thoroughly discussed, as well as how Go code in larger projects is structured (Ch. 9).

Part 3 learns you how to work with files in different formats (Ch. 12) and how to leverage the error-handling mechanism in Go (Ch. 13). It also contains a thorough treatment of Go’s crown jewel: goroutines and channels as basic technique for concurrent and multicore applications (Ch. 14). Then we discuss the networking techniques in Go and apply this to distributed and web applications (Ch. 15).

Part 4 shows you a number of Go language patterns and idioms (Ch. 16, 17), together with a collection of useful code snippets (Ch. 18). With all of the techniques which you have learned in the previous chapters, a complete Go project is built (Ch. 19) and you get an introduction in how to use Go in the cloud (Google App Engine) (Ch. 20). In the last chapter (Ch. 21) we discuss some real world uses of go in businesses and organizations all over the world.

V has very much a ‘no nonsense’ approach to it: extreme care has gone into making things easy and automatic; it adheres to the KISS principle from Agile programming: Keep It Short and Simple!

Solving or leaving out many of the ‘open’ features in C, C++ or Java makes the developer’s life much easier! A few examples are: default initializations of variables; memory is allocated and freed automatically; fewer, but more powerful control constructs. As we will see V also aims to prevent unnecessary typing: often V code is shorter and easier to read than code from the classic object-oriented languages.

V is simple enough to fit in your head, which can’t be said from C++ or Java; the barrier to entry is low, compared to e.g. Scala. V is a modern day C.

And of course this is what you must do: after setting up a V environment with a decent editor, start experimenting with the code examples and try the exercises: mastering a new language and new concepts can only be achieved through exercising and experimenting, so the text contains 130 exercises, with downloadable solutions.

Every listing is a working code example, and its output is shown after the code in // or /\* \*/ comments, or after the code lines itself as a comment (//>) when this is clearer.

The book has a website ( ), from where the code examples can be downloaded and on which complementary material and updates are available.

I would like to express my sincere gratitude to Alex Medvednikov for creating this superb language.

Welcome to the wonderful world of developing in V!

TABLE OF CONTENTS

?? these have to be checked later

QUOTES – ONE-LINERS

V is

"Go done right"

"Go for people who hate Google"

V is Go++

A perfect mix between Go and Rust; Go 4.0

C power with Go checks

Simply powerful

One language for everything

Save energy, save the planet by using V - the amount of cpu cycles we waste on the servers/clients

V programmers call themself deVelopers

PART 1 – **GETTING STARTED WITH V**

# Chapter 1 – Origins, Context and Popularity of V

## 1.1 Origins and evolution

V's year of genesis was 2017, and the date of its public launch was Jun 22 2019.

It is a new *open-source language* (MIT License), started by Alexander Medvednikov in Oct 2017.

The current (Oct 29 2019) version is 0.1.22.

### 1.1.1 Origin – Development model

Medvednikov started developing a new application called **Volt** in 2017. Volt is a fast native desktop client for all major messaging services (you can read more about it in § 1.1.8). Volt is written in the V language.

Originally the Volt app was written in Go, but after a couple of weeks of development Medvednikov decided to rewrite it in C for two reasons:

* easier integration with existing C graphics and UI libraries
* much smaller binaries

As a consequence, the app size reduced from ~5 MB in the Go version (where the Go runtime is included) to ~100 KB.

C development is not very productive, so Medvednikov spent two weeks in October 2017 to create the V language and rewrite Volt in it.

So one can say that V’s development is application-driven. Whilst V is being developed, it is used to build Volt, the V-forum, vid, etc.

Making a programming language specifically for the needs of one program, then developing that language and the program together, is an underestimated strategy. This helps test the design and apply changes from experience using it.

Community-driven: Also Medvednikov occasionally uses Twitter (<https://twitter.com/v_language>) to do a poll about a certain detail aspect of the language (like the << operator or the naming of constants). The democratic voice of the interested community of developers is then taken into account when making this language design decision.

### 1.1.2 Name

The name of the language is **V**, not "Vlang" or "V-Lang" etc, pronounced [/viː/](https://en.wikipedia.org/wiki/Help:IPA/English)

*It's a simple name that reflects the simplicity of the language, and it's easy to pronounce for everyone in the world.*

V’s source files have the extension **.v**

V header files (definition files) contain only the signature of functions and types, not their code body. They have the extension **.vh** ; theyare only used for precompiled modules.

V script files have the extension **.vsh**

Conflicts: search-websites with search term “V programming language” also lead to <http://vlang.org> (which is a verification language built on top of D) or <https://www.vcode.org/> (another general purpose programming language).

There is also Verilog, a hardware description language (see <https://en.wikipedia.org/wiki/Verilog>),

whose filenames have the same extension .v

Coq (= formal proof management system) source files are also customarily named .v

Looking at <https://github.com/github/linguist/blob/master/lib/linguist/languages.yml> .v is used by Coq and Verilog.

The search term “V programming language” gives V’s website as 1st result on DuckDuckGo and Google.

### 1.1.3 Designer

Alexander Medvednikov is a Go and C programmer, who has previously worked for Booking.com in the Netherlands. Currently he is working on Volt and V fulltime since Oct 2018.



Figure 1.1: The father of V: Alexander Medvednikov

Email:  alex@medvednikov.com

GitHub: <https://github.com/medvednikov/>

### 1.1.4 Website and communication channels

The V website is: <http://vlang.io/>

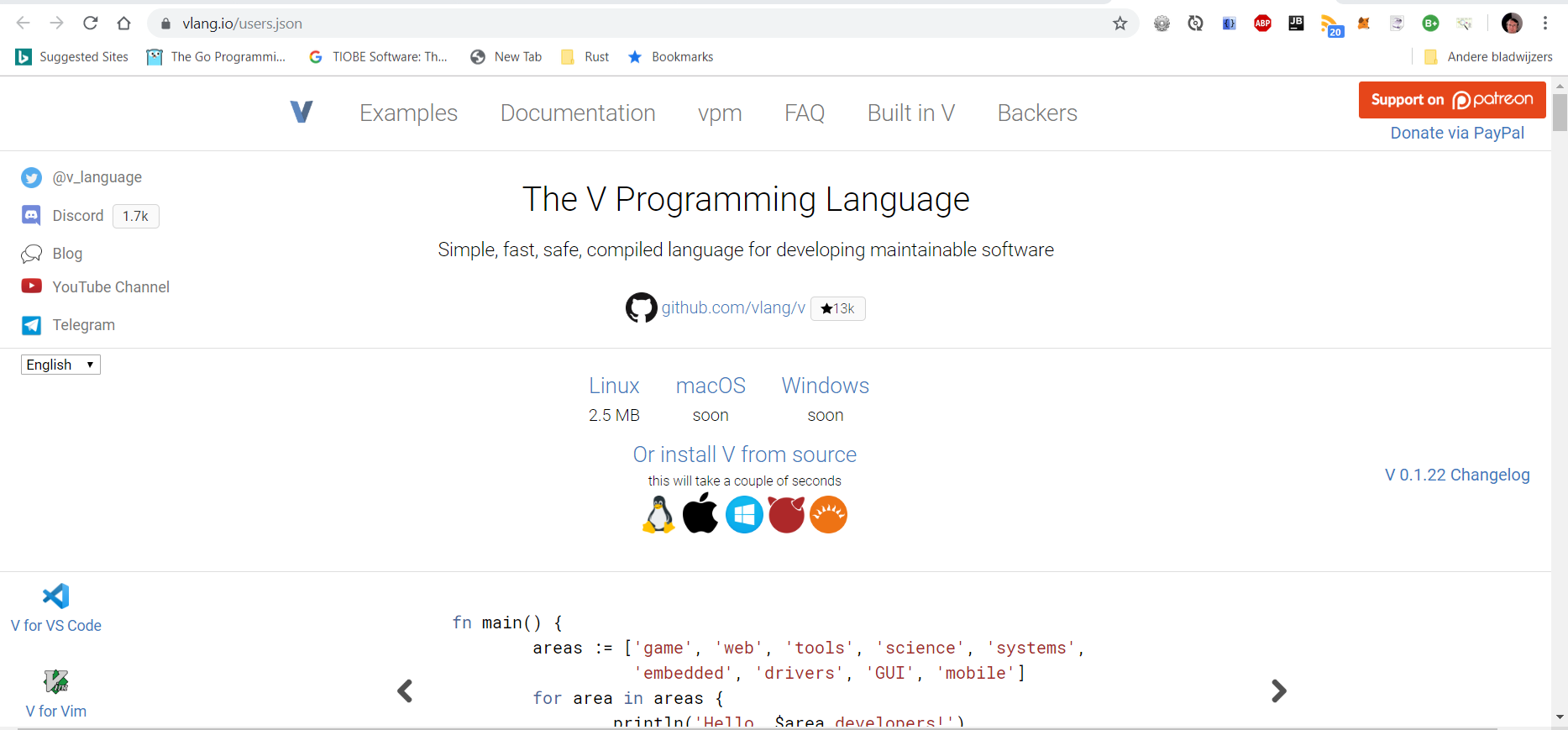


Figure 1.2: Screenshot Oct 29 2019:

The source code of the language is here: [https://github.com/vlang/](https://github.com/vlang-io/website)v

The source code of the website is here: [https://github.com/vlang/website](https://github.com/vlang-io/website)

The V documentation can be found at <https://vlang.io/docs>

Wiki: <https://github.com/vlang/v/wiki>

The Wikipedia page is at: <https://en.wikipedia.org/wiki/Draft:V_(programming_language)>

(?? Not available Oct 2019: V can be interactively tried out in your browser via the V Playground: https://vlang.io/play )

Alternative playground (runs only until version 0.1.12): <https://vlang.ide.judge0.com/>

Communication channels:

Discord : <https://discord.gg/n7c74HM>

GitHub issues: [https://github.com/vlang/V/issues](https://github.com/vlang-io/V/issues)

Reddit: [https://www.reddit.com/r/vlang/](https://old.reddit.com/r/vlang/)

([amedvednikov](https://news.ycombinator.com/user?id=amedvednikov) and u/volt\_dev is Alexander M.)

Twitter: <https://twitter.com/v_language> #vlang

Stack Overflow: Oct 29 2019: 6 questions

<https://stackoverflow.com/questions/tagged/vlang>

### 1.1.5 Logo

V has the following logo:



Figure 1.3: V’s logo

### 1.1.6 History

*Age: 2 years*

* Development started in **October 2017**; the first V compiler is written in Go

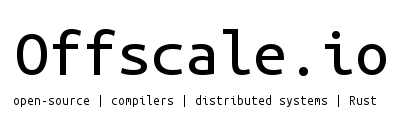
**2018**

* Starting from Feb A.M. has been working full-time on V and Volt.
* Mar: V compiler is self hosting (written in V)

**2019**

* Feb 7: first public release of Volt v0.35 for macOS
* Feb 7: First article mention in Reddit/Go:

<https://www.reddit.com/r/golang/comments/ao0iar/originally_volt_app_was_written_in_go_but_after_a/>

* Feb 18: machine code generation and the C++ translator are in early stages
* Feb 21: first version of V website [vlang.io](https://vlang.io)
* Mar 8: Volt 1st version for Windows (v 0.80) has been released
* Mar 24: The blog is live and it’s the first public web app written in V: https://blog.vlang.io/
* Mar 28: Volt 1st version for Linux has been released
* **Mar 30**: PlayGround opened <https://vlang.io/play>
* Apr 12: V forum is live
* **Apr 15**: Early access to V compiler for macOS (**v 0.0.1**) for Patreon 30$ supporters
* Apr 23: Same for Linux (**v 0.0.2**): the compiler updates automatically.
* Apr 26: **v 0.0.4** for Linux and macOS
* May 3: **v 0.0.7** for Linux and macOS
* May 18: V got the first sponsor: (Nov: ??)[](https://offscale.io/)
* May 24: **v 0.0.9**

Changelog:

ui module (not on Linux)

- native http.v on Windows without libcurl

- webview module

- `foo[i].bar` syntax

- `a().b().c` syntax

- lots of vfmt fixes

- fixed size arrays (`[100]byte`)

- high order functions (`fn foo(a fn(int) int) fn(string) {`)

- improved testing

- `a..b` ranges

- cleaned up types (no more int64 etc)

- all C code removed from vlib (replaced with V)

- major memory leaks fixed in the `gg` module (opengl textures and text rendering)

- operator overloading

- structs and other types can now be declared after usage

- `val in array`, `if num in [1,2,3] {`

- optimized strings (using a pool instead of allocations for small strings)

* May 28: **v 0.0.10**
* **Jun 22**: **open-source release v 0.0.12**

June 22 signifies a special date for V:  
V is the 22nd letter in the alphabet  
V is also one of the symmetrical letters  
June is mid year  
18:00 UTC = 6 p.m.

* Jun 24: binary release of v 0.0.12 for macOS and Linux
  + Ui examples don’t work because ui library does not exist

**V 0.1.7:**

- All C code in the compiler and vlib has been replaced with V.

- `#` syntax for embedding C code has been removed.

- Exported functions now need to be marked with `pub`, all public vlib functions have been updated.

- CI has been set up (Travis + Azure). On every commit and PR it is made sure that V

can compile itself, all tests pass, and all examples compile.

- More tests have been uploaded.

- Cleaner bytes to string conversion: `tos2(bytes)` => `string(bytes)`.

- The home page has 3 more examples next to 'hello world' that show the features of the language.

- Lots of bugs and issues fixed.

* Jun 27: **V 0.1.8** – macOS + Linux
* Single file programs without `fn main` now work as expected.
* REPL has been fixed: it now supports imports, consts, function definitions, etc.
* **Jun 29**: **V 0.1.9 / V 0.1.10** – macOS/Linux/Windows

- **Windows support via MinGW-w64. Pre-built Windows binary.**

- File structure has been simplified: all vlib modules were moved to the vlib/ directory,

makefile was moved to the root.

- One single archive with pre-built binaries for all operating systems.

- `mut var := val` was fixed (previously `mut var = val` was allowed as well).

* Jul 1: **V 0.1.11** – macOS/Linux/Windows

- Cross compilation for Windows

- Lots of Windows fixes

- socket.v

- maps fixed

Jul 4: **V 0.1.12** – macOS/Linux (no Windows binary)

* **V can finally compile itself on Windows**. (https://github.com/vlang/v#mingw-w64)
* `os` module now uses optionals in all functions that return `File`. Lots of bugs with optionals fixed.
* `println` was optimized. It no longer results in allocations. Now it also works correctly with all integer types.
* Lots of `vfmt` fixes, it will be enabled tomorrow.
* New `strings` module.
* Lots of other fixes and improvements, thanks to all the contributors.
* **Jul 8**: talk by Alex M on the V programming language in Tilburg, Netherlands

<https://www.meetup.com/Linux-Open-Source-Tilburg/events/262250468/>

Jul 10: **V 0.1.13** – macOS/Linux/Windows

* New enum syntax (token == .name), enum values are no longer global consts
* Submodules (import encoding.base64)
* Hot code reloading
* Special err variable for getting error values
* Complex numbers –
* << can now append arrays (numbers << [1, 2, 3])
* Lots of Windows fixes (Windows still needs some work)
* Lots of REPL improvements (e.g. >> 2 + 3 works now, no println required)

Jul 12: **V 0.1.14** – macOS/Linux/Windows

* gg module Windows support
* V Tetris runs on Windows.
* glad and cJSON are now compiled only once, this makes compilation of programs using gg and `json a bit faster.
* v.c has been cleaned up and minimized (~16k => ~10k lines of code)
* type aliases can now have methods.
* Const overflow check during compilation (byte(1000) will no longer compile)

Jul 15: **V 0.1.15**:

- FreeBSD, OpenBSD, NetBSD, DragonFly support.

- Hot code reloading now works with graphical applications: <https://github.com/vlang/v/blob/master/examples/hot_code_reloading/bounce.v>

- VROOT was removed, the installation process is now much simpler.

- defer statement

- map.v was re-written. It's now much faster. - for key, val in map syntax.

- flag module for parsing command line arguments.

- zip module.

- crypto/sha1 module.

- Submodules and module aliases (import encoding.base64 as b64).

Jul 23 - **V 0.1.16**:

* V can now be used with Visual Studio!
* Hot code reloading now works with graphical applications (e.g. graph.v, bounce.v).
* Compile time memory management for arrays.
* High order functions.
* match expression (replacing switch).
* Import cycle detection.
* crypto/md5, crypto/sha256, and crypro/sha512 modules.
* os.executable() - a cross platform function that returns full path to current executable.
* ~/.vlang and VROOT were removed entirely. The installation is a lot cleaner now.
* V can now be packaged for all Linux distros.
* Arch Linux package.
* string(bytes\_buffer, len), string(bytes\_array) casts.
* Multiple defers.
* key in map syntax (replacing map.exists(key)).

**Jul 29 - V 0.1.17**

- vweb module for developing web apps in V.

- vtalk, open source V forum software.

- Generics (very limited right now, but they will be gradually improved).  
- Comptime codegen (foo.$method() where method is a string).

- @ for escaping keywords (e.g. struct Foo { @type string }).

- Windows Unicode fixes (V can now work with non-ASCII paths etc on Windows).

- Fix mutable args bugs + don't allow primitive arguments to be modified.  
- Declaring a mutable variable and never modifying it results in a compilation error.

- Interactive debugging support.

- sync module for Windows. - #! support on Unix systems (V scripts).

- Lots of Visual Studio fixes.

- crypto.aes and crypto.rc4 modules.

- Internal modules.

* **Aug 8:** open-source release of vid
* **Aug 16: V 0.1.18 –**
* Built-in ORM (uk\_customers = db.select from Customer where country == 'uk' && nr\_orders > 0)
* Map initialization syntax: m := { ‘foo’: ‘bar’, ‘baz’: ‘bar’ }
* map.delete(key).
* libcurl dependency was removed from the http module.
* All function arguments are now immutable by default (previously they could be modifed inside the function).
* http functions now return optionals.
* sync.WaitGroup.
* vweb static files serving.
* crypto.rand module.
* v up to update V.
* SChannel support on Windows.
* net.urllib module.
* vpm package manager, v install.
* () are now required in complex bool expressions: (a && b) || c instead of a && b || c
* All arrays now have a default.str()method.
* Bootstrapping V with MSVC.
* Experimental≠etc support.
* encoding.csv module.
* $if debug {for running code in debug mode only.
* Map struct fields are now initialized automatically, just like arrays.
* Maps now support array values.
* json functions can no longer be used if the json` module is not imported.

**V 0.1.19 - Sep 12 2019**

- Lots of refactoring, simplifications, and optimizations in the compiler.

- Experimental memory management at compilation (only for the V compiler itself for now).

- Lots of ORM fixes.

- Functions can now be inlined via the [inline] attribute.

- New mysql module.

- Better error format that is supported by all major editors (go to error).

- Error messages now point to the actual place where the error happened.

- Custom json field names: struct User { last\_name string [json:lastName] }.

- Raw json fields via the [raw] attribute.

- import const was removed from the language.

- All C code was removed from the freetype module.

- gg module can now render all Unicode characters.

- [typedef] attribute for imported C struct typedefs.

- Support of Objective C interfaces (primarily for using Cocoa).

- REPL: clear command and custom functions.

- REPL tests (which are also used for testing certain compiler errors).

- Syntax bug fixed: foo[0] += 10 is now possible.

- http: support plain HTTP protocol and follow redirects.

- http: header data is now processed correctly.

- net: basic UDP support.

- array.contains() was removed from the language (in should be used instead).

- [0; len] syntax was removed (replaced with a simpler [0].repeat(len))

- Primitive aliases were removed to simplify the language.

- GitHub supports V now!

- Backtraces are now printed on panics.

- A new awesome readline module.

- V.c is now regenerated automatically after every commit.

- A bug with struct ordering was fixed, now structs can be declared in any order.

- V modules can now be built with v build module.

- @FILE, @LINE, @FN, @COLUMN for debugging.

- JavaScript backend! (big project, WIP until Sep 13)

**V 0.1.20 - Sep 17 2019**

- JavaScript backend!

- Hundreds of C warnings were fixed. `gcc v.c` now builds without

any warnings.

- The mutability check now applies to function args (mutable

receivers that are not modified result in a compilation error).

- V tests now show how long each test took.

- Official Android support (only console applications via Termux for now).

- Typo check. If a variable/function/module etc is misspelled,

V will suggest the correct name.

- Lots of Microsoft C fixes, and a separate Travis instance for

**V 0.1.21 - Sep 30 2019**

* none keyword for optionals.
* Solaris support.
* All table lookup functions now use none.
* varargs: fn foo(bar int, params ...string) {
* Double quotes (") can now also be used to denote strings.
* GitHub Actions CI in addition to Travis.
* -compress option. The V binary built with -compress is only ~90 KB!
* More memory management.
* Unused modules result in an error.
* "Unused variable/module" errors are now warnings in non-production builds.
* Duplicate methods with the same name can no longer be defined.
* Struct names must be capitalized, variable/function names must use snakecase.
* Error messages are now even nicer!
* Lots of fixes in automatic *.str()* method generation for structs and arrays.
* ~30% faster parser (files are no longer parsed separately for each pass).
* \_ is no longer a variable, but an actual syntax construct to skip unused values, like in Go.
* Multiple returns (fn foo() (int, string) {).
* ! can now only be used with booleans.

**V 0.1.22** **28 Oct 2019**

- Generic functions (fn foo<T>(bar T) T {) with varargs support.

- array[start..end] and string[start..end] slicing syntax.

- Optimized array.filter() and array.map().

- sqlite module.

- Cached modules for faster compilation.

- Dramatic compilation optimizations: V now compiles itself in 0.10 - 0.30 seconds.

- V scripts (simpler and cross-platform alternative to Bash).

- Infinite multi-dimensional arrays ([][][]int).

- unsafe.

- [deprecated] attribute.

- [if] function attributes for compile time function exclusion for performance.

- switch has been completely removed from the language and replaced by match everywhere.

- pub struct and pub const, previously all structs and consts were public by default.

- musl support (V can now run on, for example, Alpine Linux).

- Module header generation. V now supports closed source modules, which are still used in some industries.

- Constants were added to typo suggestions.

- color in [.green, .red, .blue] now works without specifying Color.green.

- V compiler is now a module that can be used by other programs.

- Backtraces now have source lines on Linux.

- runtime.nr\_cpus().

- fn init() for module initialization.

- a in [1, 2, 3] optimization: no array gets allocated.

- Raw strings: s := r'hello\nworld'.

- if a := func() { } syntax for handling optionals.

- f32/f64 comparison now uses machine epsilon by default.

**V 0.1.23** **30 Nov 2019**

The last release before 0.2.

- Direct x64 machine code generation. Hello world being built in 3 milliseconds.

- Bare metal support via the `-freestanding` flag, allowing to build programs without linking to libc.

- Prebuilt V packages for Linux, macOS, and Windows.

- `string.index()` now returns `?int` instead of `int/-1`.

- Lots of fixes in Generics.

- vweb framework for developing web applications is back.

- Vorum, the forum/blogging software written in V/vweb, can now be compiled and has been added to CI.

- REPL, `v up` have been split up into separate applications to keep the core V compiler small.

- V now enforces short enum syntax (`.green` instead of `Color.green`) when it's enough.

- V UI for macOS.

- Interfaces have been rewritten. `[]interface` support.

- `os.cp()` for copying files and directores.

- Additional compile-time flags: `$if clang, msvc, mingw, x32, x64, big\_endian, little\_endian {`.

- All C functions now have to be declared, all missing C functions have been defined.

- Global variables (only with the `--enable-globals` flag) for low level applications like kernels and drivers.

- Nothing can be cast to bool (previously code like `if bool(1) {` worked.

- `<<` and `>>` now work with all integer types.

- V detects Cygwin and shows an error. (V supports Windows natively)

- Improved type checking of some operators (`%, |, &` etc).

- Windows 7 support.

- `println(true)` now prints `true` instead of `1`.

- `os.exec()` now uses `CreateProcess` on Windows.

- fast.vlang.io website for monitoring the performance of V after every commit.

- On Windows Visual Studio is now used automatically if GCC is not installed.

- vfmt!

- Lots of cleaning up in the compiler code.

- Multi-level pointers in unsafe code (`\*\*\*\*int`).

- MSVC backtrace.

- `$if os {` blocks are now skipped on a different OS.

- C string literals (`c'hello'`).

- AlpineLinux/musl fixes + added to CI.

- Inline assembly.

- Clipboard module (Windows, macOS, X).

- `foo()?` syntax for error propagation.

- Docs have been migrated from HTML to `doc/docs.md`.

- `eventbus` module.

- Haiku OS support.

- `malloc/free` on bare metal.

- `utf8` helper functions (`to\_lower()`, `to\_upper()`, etc).

- Optimization of `for c in str {`.

- `string/array.left/right/slice/substr` were removed (`[a..b]` slicing syntax should be used instead).

The growth of V can be followed in an animation: <https://www.youtube.com/watch?v=yJCKGIUfIP4>

Roadmap:

A project roadmap for V 0.2 can be followed here: <https://github.com/vlang/v/projects/2>

An alpha release (v 0.2) is planned for Nov 18 2019 .

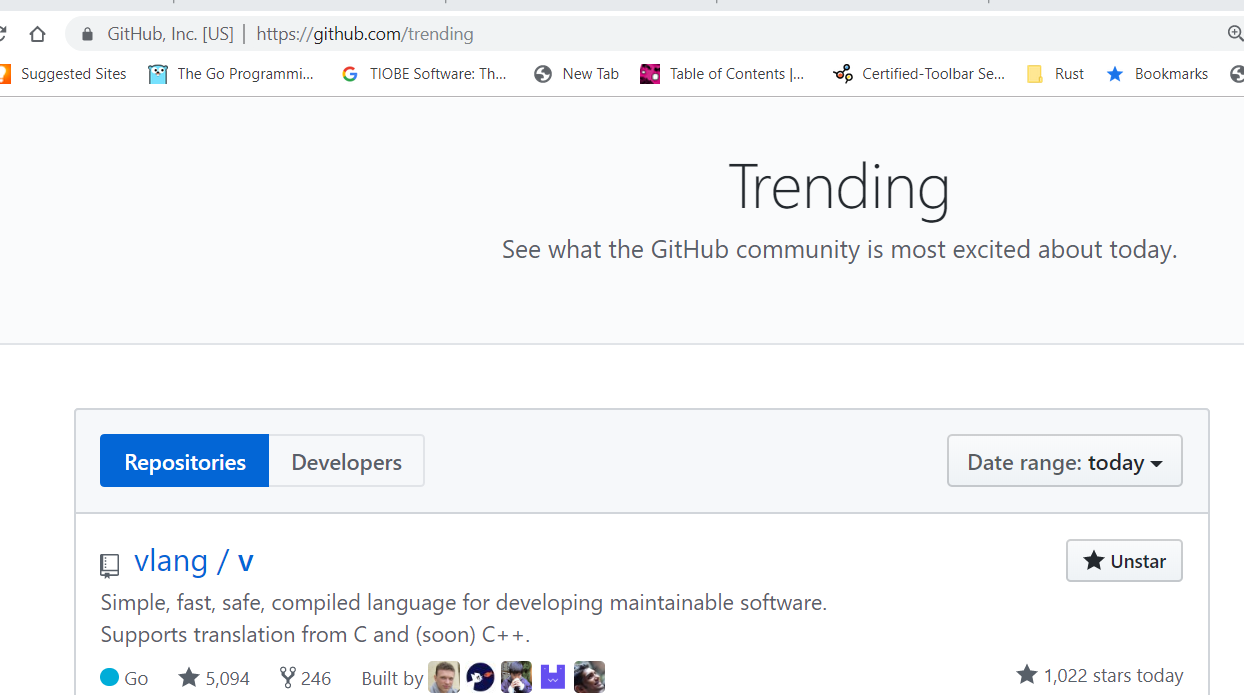
A release candidate is planned for Dec 23 2019 .

The first production-ready version (v 1.0) will be published on Jan 6 2020.

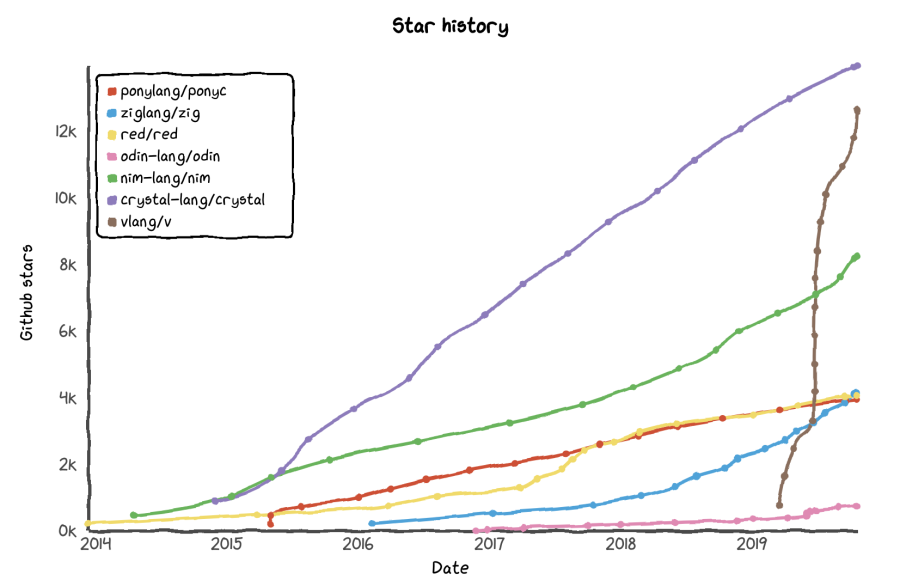
### 1.1.7 Popularity

For numbers of followers and supporters, see *v\_github.xlsx*

V appears regularly in top trending GitHub repository: for example Jun 24



GitHub stars evolution:



### 1.1.8 The Volt app

As already mentioned (§ 1.1.1), Volt (previously called Eul) was the driving force for the start of the V-language development.

Volt was built to solve two big problems with the current IM (instant messaging) solutions: there are too many of them, and the clients are ridiculously bloated for what they are built to do.  
Volt is going to be a simple and fast hub to connect to all your contacts across multiple services (similar to what Trillian and Miranda did in the past).

The Volt website is https://volt-app.com

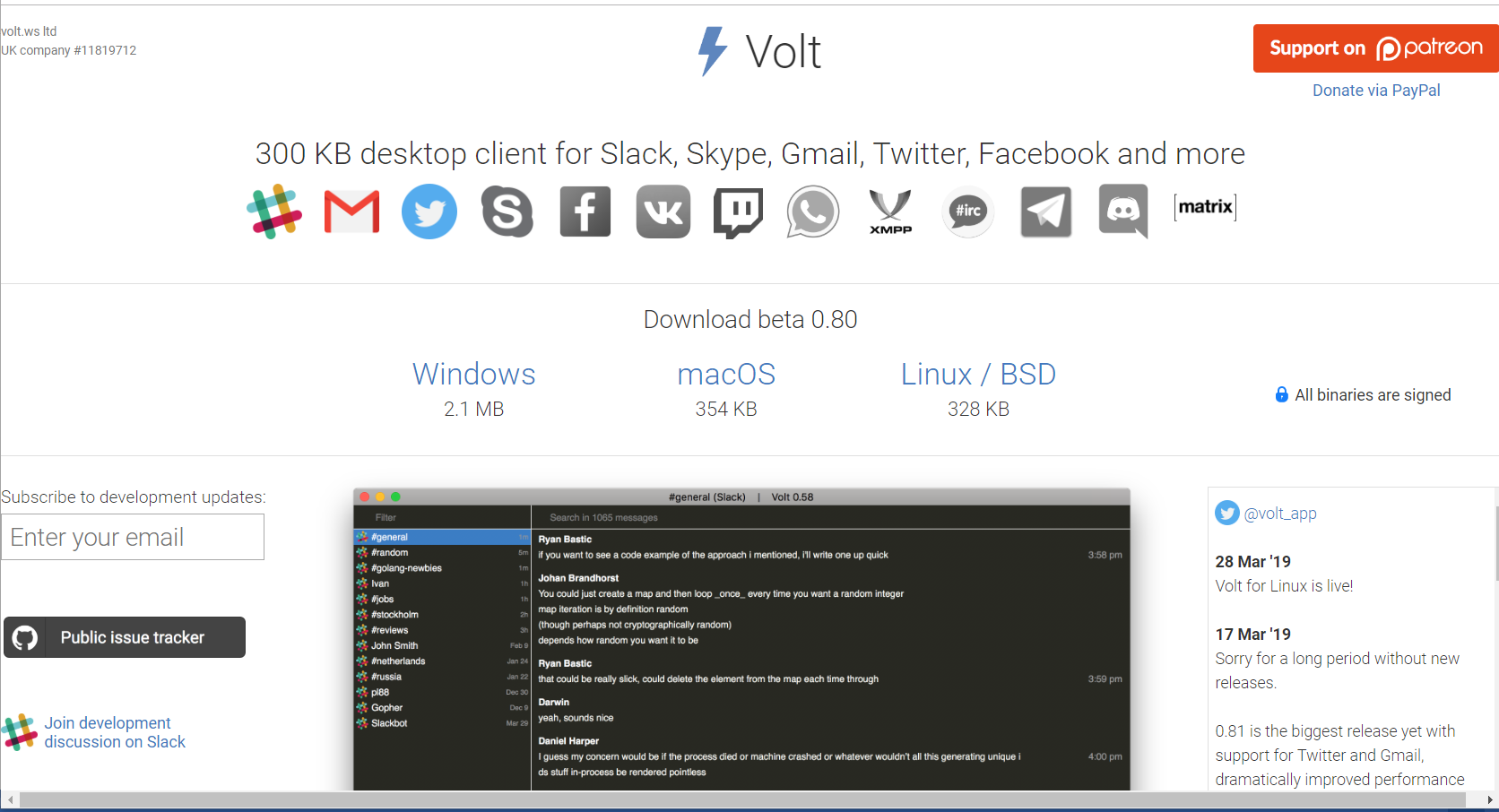


Figure 1.4: Screenshot Mar 31 2019:

Right now it has basic support for Skype, Slack, Gmail, and Facebook. More protocols are coming soon: Discord, Skype for Business, XMPP (Jabber), IRC.  
Volt is blazing fast. Unlike official clients, it can handle hundreds of thousands of messages in one chat without lag. And it's size is only about 300 KB, with minimal CPU and RAM usage.  
  
Communication channels:

Slack: <https://voltapp.slack.com>

Twitter: <https://twitter.com/volt_app>

Email: [support@volt.ws](mailto:support@volt.ws)

### 1.1.9 Business and support model

Both Volt and the development of V are crowd-funded via Patreon:

V : <https://www.patreon.com/vlang/overview>

Volt : <https://www.patreon.com/voltapp>

Volt exists as a UK company: **volt.ws** ltd UK company #11819712

Volt: The app is always going to be **free.**  Hopefully as Volt grows, it will get more supporters on Patreon. There will also be an affordable premium plan for things like having more than 5 accounts. The base functionality will always be free.There will be no ads and no datatracking**.**

AM hopes for V sponsor companies to support development on the longer term.

## 1.2 Main characteristics, context and reasons for developing a new language

**1.2.1 Languages that influenced V**

V belongs to the C-family, like C++, Java, C# and Go, and it is inspired by several languages created and used by its designer. It is a very small language, smaller than C.

It is very similar to *Go*, from which it takes much of its syntax and semantics. It was also influenced by *Oberon*, *Rust* (memory management system) and *Swift*. It supports the same paradigms as Go.

Compared to Go, it’s syntax is even more simplified and made more concise and clean. It also has characteristics of a dynamic language, so Python and Ruby programmers feel more comfortable with it.

It is an open-source language, distributed with a **MIT license**, so it can be used by everybody, even for commercial purposes without a fee, and it can also be changed by others.

**1.2.2 Why a new language?**

“The goal of V is to be a bridge between statically compiled and interpreted languages.

Many people often do prototyping or quick small programs in Python.

I hope to make V suitable for that.”

- Before V and Go, a developer had to choose between fast execution but slow and inefficient building (like C++), efficient compilation but not so fast execution (like .NET or Java), or ease of programming but slower execution (like dynamic languages such as Python, Ruby or JavaScript): V is an attempt to combine all three wishes: *efficient and fast compilation, fast execution, ease of programming*.

- Also V has *small sized executables*.

According to the designer he created V because none of the existing languages had all of the following features:

|  |  |
| --- | --- |
| Fast compilation | Go, Delphi |
| Simplicity | Go, Python |
| Great performance on par with C (within 3%) | C, C++, Delphi, Rust, Nim |
| Safety (immutability, no null, option types, free from data races) | Rust |
| Easy concurrency | Go |
| Zero cost C interop | C, C++, Nim |

**1.2.3 Targets of the language: V is a fast, safe and compiled language**

A main target was to combine the *efficacy, speed and safety* of a *strongly and statically compiled language* with the *ease of programming* of a *dynamic* language, so as to make programming more fun again.

So the language is *type-safe*, and it is also *memory-safe*: pointers are used in V, but pointer-arithmetic is not possible.

V is designed for building maintainable and predictable software

The frustration with the exorbitant build-times of C++-projects in the industry is well known. Like Go, of the utmost importance for V was also the *building* *speed (compilation and linking to produce machine code)*, which had to be excellent (in the order of ns to a few s at most). This alone should give an enormous boost to *developer productivity* and give rise to a tighter code and test development cycle.

Typical V projects compile in half a second: this lightning fast compiling process, even faster than C or Fortran, makes compilation a non-issue. Until now this was regarded as one of the great benefits of dynamic languages because the long compile/link step of C++ could be skipped, but with V this is no longer an issue. Compilation times are negligible, so with V we have the same productivity as in the development cycle of a scripting or dynamic language.

On the other hand, the *execution speed* of the native code is comparable to C/C++.

Furthermore, there is excellent *support for legacy software*, because of the ease of interaction with C (and C++): see § 3.9.

### 1.2.4 Guiding design priorities

The core design principles of V are memory safety, C/C++ interaction fluency , and developer efficiency, fast compilation time, simplicity, maintainability.

“V is more expressive and easier to understand so it is therefore much easier to maintain. It is a high level language that compiles and runs with the efficiency of a low level language. The Holy Grail to be sure.”

A. **Simplicity**: a very light and minimalistic language, drawing much inspiration from Go.

V tries to reduce typing, clutter and complexity in coding through a minimal amount of syntactical tokens and keywords (22)

This, together with the *clean, regular and concise syntax* and *mandatory formatting*, enhances the compilation speed, because the keywords can be parsed without a symbol table.

For example: being forced to place `{` on the same line and using tabs instead of spaces. This in the end saves a lot of time, avoids arguments on code styles and on which paradigm to use, and makes code simpler. Of course, this formatting needs to be done automatically through the **vfmt** tool, which should be applied when a source file is saved and before being compiled.

Examples of this aim for simplicity are:

* using the keyword fn for functions, instead of func as in Go
* using u64 instead of uint64
* there is no ‘:’ between a name and its type
* there is no ‘–>’ before the return type of a function
* there is no ‘,’ between fields in structs

These aspects reduce the number of code lines necessary, even compared with a language like Java.

V has a *minimalist approach*: there tends to be *only one way of doing things*, so reading other people’s code is generally pretty easy.

B. **Readability**

This is an immediate consequence of target A. All developers know *readability of code* is of the utmost importance in software engineering, because most of a project’s time is spent in the maintenance phase.

V supports only one way of doing things. It is also a very strict language, which restricts the developer’s freedom in a certain way, in contrast to C++, Nim, Rust, Scala etc. which are languages that support lots of paradigms and give developers lots of freedom. But the result of more freedom is complexity increase, code can get hard to understand, and maintainability suffers.

You will be able to jump right in any part of a large code base and feel like it was you who wrote the code.

The design concepts of the language don’t stand in each other’s way, they don’t add up complexity to one another: they are *orthogonal.*

C. **Safety**

(Where V improves over and differs from Go, this is indicated with >< Go)

1) *No global variables, global state is not allowed*. (>< Go)

However constants can be defined at the global level with the const keyword (see ch 4 § 4.3 ).

Variables can only be defined within functions, also in fn main()

2) *Variables are immutable by default*. (>< Go)

A mutable variable must be explicitly indicated with the mut keyword.

3) *Variables are automatically initialized to empty (zero) values.*

Undefined values cannot occur.

*4) There's no null or nil value, so no more null reference crashes can occur.* (>< Go)

Working with nil values is replaced by **option types**, as in C# or Rust (see ch 12).

You can't have a null pointer in V, since they are required to be initialized by the compiler: a := &b.

If you have a struct field with a pointer, and it's not initialized during creation, the compiler will automatically allocate memory for it.

V can work with C code, of course C has null pointers, so you can get null bugs there, just like in any language calling C code.

5) *Functions are partially pure.* (>< Go)

Function arguments are by default immutable, if they should be changed, it must be indicated with mut; only the method's receiver can be modified.

6) *Thread safety and guaranteed absence of data races*, with no performance costs.

For example: if you are using a hash map in a concurrent function, a thread safe hash map is used automatically. Otherwise a faster single thread hash map is used.

7) *Automatic memory release*: no need to manually free the memory either! V's memory management is similar to Rust, but it's much simpler. V manages memory at compilation time (like Rust).

How can V combine heap allocations with no garbage collection (e.g. Go), no manual memory management (e.g. C[++]), and no typing system to ensure lifetimes (e.g. Rust)?

Basically the V compiler will analyze your code and insert necessary free() calls during compilation: a variable is freed at the end of the scope in which it was defined.

fn foo() {

s := 'hi'

println(s)

}

* after the } s will be freed automatically.

That’s a reason why globals are forbidden, because they always remain in (global) scope, and their memory can never be freed.

Mar 2019: Right now only basic cases are handled. For others, manual memory management is required for now. The right approach to solve this will be figured out in the near future.

Apr 25: There will be a way to replace the standard libraries heap allocator

May 6: You’ll be able to provide a custom allocator, and/or I’ll add an option to disable allocation ( -noalloc)

Only objects declared with & are allocated on the heap.

8)  *All exported fields are read-only by default in V*. (>< Go)

9) *No default arguments and no overloading.*

D. **Fast compilation** (see chapter 2)

E. **High performance** (see § 3.7)

Execution is fast because V compiles to native code, either directly or through first compiling to C and then going through an optimizing C compiler. Performance is as fast as C.

V has no garbage collector, so execution is not slowed down by this process.

V's standard library was built with a focus on performance as well. Also V has no (or a very small) runtime.

Examples:

* + Because strings are immutable, creating substrings never results in allocations and copying of data
  + The number of memory allocations is minimal.
  + Small strings are allocated on the stack
  + Serialization is built in and doesn't use reflection so it's an order of magnitude faster than most implementations in other languages.

**1.2.5 Characteristics of the language**

V is, like Go, essentially an *imperative* (procedural, structural) kind of language, built with *concurrency* in mind.

It is *not object-oriented* in the normal sense like Java and C++, because it doesn’t have the concept of classes and inheritance. Like Go, V has structs and embedding, which can simulate inheritance. V also does have a concept of *interfaces*, with which much of polymorphism can be realized. V has a clear and expressive *type-system*, but it is lightweight and without hierarchy. So in this respect it could be called a hybrid language.

Object-orientation as in the predominant OO-languages was considered to be too ‘heavy’, leading to often cumbersome development constructing big type-hierarchies, and so not compliant with the speed goal of the language.

Functions are the basic building blocks in V, and they are first class citizens in the language: their use is very versatile.

In chapter 6, we will see that V also exhibits the fundamental aspects of a *functional language*.

V is *statically typed*, so it is a *safe language*, and it compiles to native code, so it has a very efficient execution. It is *strongly typed*: implicit type conversions (also called castings or coercions) are not allowed; the principle is: keep things explicit!

It has certain characteristics of a *dynamically* typed language through the *automatic type-inference with the := operator*.

That’s why it also appeals to programmers who left Java and the .Net world for Python, Ruby, PHP and JavaScript.

There are no exceptions, only Go-like panics.

V uses *UTF-8* throughout, not only in strings, but also in program code (the source-code file encoding is UTF-8): so V is truly international.

?? Apr 23: Compile-time reflection will be built-in, so that everybody will be able to generate code at compilation time with a very simple syntax.

**1.2.6 Language comparisons**

#### 1.2.6.1 Comparison with Go

1) *No verbose and sometimes unsafe error checking*: if err != nil checks like in Go. In V, these are replaced by the use of option types and the or keyword (see ch 12 ).

2) Go has a heavy runtime and that’s why it produces relatively large executables. V has *a much smaller runtime*, which results in *much smaller binaries*. For example: a simple web server written in V is around 140 KB vs 11 MB in Go.

3) V also has *no garbage collection*, resulting in top execution speed, with no slow-downs.

4) Go has not so great C interop with cgo, V has a much *simpler and zero-cost C interop* (see § 3.9)

5) Go has 2 declaration styles: with the var keyword and subsequent initialization, and the := operator with type inference. V makes this simpler by allowing *only one declaration style* (a := 42).

6) As of Go 1.12, binaries are dynamically linked to libc on macOS, Solaris, and Windows. If you use networking, libc is used on Linux as well. V uses libc, but only links to it statically.

7) In Go reflection is implemented in runtime. It's a huge cause of binary size bloat (info for every type has to be stored), and actually programming with the reflect package is a pain. Worst is that it's used by all the encoding packages in the standard library, so it's hard to avoid. In V,  compile-time reflection for JSON encoding is used.  
  
6) V has *much stricter code formatting* tools, to ensure one coding style. This is implemented in its vfmt tool, compared to the gofmt tool of Go.

7) V already has *generics,*  while Go doesn’t have it (it will perhaps be implemented in Go 2.0)

8) V has *cheaper interfaces without dynamic dispatch* compared to Go. There’s no dynamic dispatch in the language at all, even for interfaces.

9) V has *fearless concurrency*, there is a guarantee at compilation that there will be no data race.

10) V has s*tring interpolation*:  println('$foo: $bar.baz')

A detailed comparison with other languages can be found here: <http://vlang.io/compare>

#### 1.2.6.2 Comparison with Zig

V improves upon Zig on the following characteristics:

* Simplicity
* Strings
* No globals
* Easier allocations
* Automatic memory management
* No LLVM dependency
* Hot reloading (<https://github.com/vlang/v/issues/30>)
* [Faster compilation](https://vlang.io/compilation_speed)

**1.2.7 Use cases of the language**

Languages that can [emit C](https://archive.fo/psDOV#selection-583.1-532.18) and [easily wrap existing C libs](https://archive.fo/z1JJu#selection-1379.55-1379.76) like Vlang and Nim are "general purpose" in the fullest sense of the term. There are C/C++ libraries for everything: kernel modules, GUI, games, [Web frameworks](https://www.techempower.com/benchmarks/#section=data-r17&hw=ph&test=plaintext&l=ziimf2-1), mobile, embedded and server databases, big data, high-level networking, [JIT](https://softwareengineering.stackexchange.com/questions/29344/jit-compiler-for-c-c-and-the-likes), AI, [etc](https://github.com/kozross/awesome-c).

V is a general programming language, useful for solving text-processing problems, making frontends, or even scripting-like applications. Anything you can do with C or C++, you can do with V.

V is suitable for developing large and maintainable projects:

* + *Games*: Many game developers come to the conclusion that C++ is too complex a language. Because of its very fast compilation and live code reloading V is especially useful in game development. V can also be used as the glue or scripting language in your game projects
  + *Web services*: the Vweb (vweb)
  + *Mobile:* a major priority. Unlike Flutter, V will have a framework that uses native components, just like V ui for desktop.
  + *Native apps*: V can be used to build native apps that look native. You no longer need to embed a browser to develop cross platform apps quickly.
  + *Scientific applications*
  + *Real-time critical applications:* V offers more safety than Go, and runs without a garbage collector. That’s why it is more appropriate for this kind of applications.
  + *Systems programming:* V supports pointers, as well as everything necessary.
  + *Embedded and kernel devlelopment:* see § 13.1

V is also great as a scripting language (see ch 14):

You no longer need a separate scripting language in your projects, so no need to embed Lua or Python, learn other languages, and switch between two languages when writing your game or other complex project.

Large C/C++ projects can slowly transition to V:

Because of the easy interoperability with C/C++ , a V-project can simply using existing C/C++ libraries. Moreover, existing C/C++ apps and libraries can be transformed into V through the V translator.

**A remark about programming in V**

If you come to V and have a background in other contemporary (mostly class or inheritance-oriented languages like Java, C#, Objective C, Python or Ruby) you can fall in the trap of trying to program in V like you did in your previous language. V is built on a different model, so trying to move code from language X to V usually produces non-idiomatic code and overall works poorly: you have to start over, thinking in V.

If you take a higher point of view and start analyzing the problem from within the V mindset, often a different approach suggests itself which leads to an elegant and idiomatic V solution.

**1.2.8 Future additions and changes to V**

(these sections will have to be placed in the chapters where they belong, as soon as they are supported)

X 1) Strong modular system:

X 2) Built-in testing:

X 3) Generics :

X 4) Centralised package manager:

V has a centralized package manager with versioning, similar to Ruby gems. So no more import “github.com/…” like in Go that breaks everything on an update.

installing modules is as easy as: v install sqlite

with a centralised system it’s very easy to look up, like https://rubygems.org/

Developer has to choose a unique name (perhaps there will be `user/name`) and provide a link to the git repo.

Aug 4: V now has a package manager called **vpm**: <https://vpm.best/>

5) Concurrency, parallellism and coroutines:

AM wants to implement something similar to goroutines and the Go scheduler: timing ??

Feb 20 2019: Any plans for lightweight threads (similar to "goroutines")? As for goroutines, yes, I’m not sure about the implementation details yet. For now *V has traditional threads only, but with automatic locking to prevent data races. V will not compile your program, if data accessed from multiple coroutines is not locked*.

Mar 9 2019:

Because there are no globals, and functions can’t modify their arguments, there are *only 2 ways to modify data concurrently:*

1. Closures

mut a := 0

go fn() {

lock { a++ }

}()

2. Receivers

mut foo := Foo{}

go foo.modify()

fn (foo mut Foo) modify() {

lock {

foo.bar++

}

}

Mar 10: go f() simply runs f in a new OS-thread. So it’s very inefficient. I’m most likely going to implement real coroutines and a scheduler similar to Go.

6) WebAssembly (WASM):

Compilation to WASM will be implemented: > 2019

At this moment, you can already do: V => C => WASM.

7) Incremental builds: this is not necessary, because of the very fast compilation.

8) Compile time evaluation: Mar 19: not yet

Apr 9: V will have strong compilation time capabilities soon after open-source release.

Jul 30: simple form of codegen

9) Versions for Android and iOS in 2019 (Mar 23)

10) ARM support: will come later

X11) \*BSD support

12) Optimizer for the V compiler

X 13) Other platforms and architectures

14) Automatic memory management (like Rust): Mar 2019: near future

16) LSP (Language Server protocol) should be supported (community effort)

17) Go to V translator (perhaps)

18) Support for pattern-matching

19) Support for lambdas

X 20) Functional programming helpers to transform list (like map, filter, reduce, flatMap, etc.)

syntax will be users.filter(\_.name.starts\_with('A'))

**1.2.9 Software projects using V**

1) **V language**: the V compiler and standard library are written in V.

2) **Volt** ([**https://volt-app.com/**](https://volt-app.com/)): native desktop client for Slack, Skype, Matrix, Telegram, Twitch and many more services.

3) **Filey**: Cross platform file manager with Miller Columns and built-in selective synchronization with major cloud platforms.

4) **Vid**: Cross-platform fast open source editor with size 200 Kb and the performance of Sublime Text.

(<https://github.com/medvednikov/vid>)

5) **C/C++ to V translator**: This complex tool supports the latest standard of notoriously complex C++ and allows full automatic conversion to human readable code.

6) **V ui**: Cross platform widget toolkit using native APIs.

7) **gitly** (<http://gitly.org/>): an open-source GitHub/GitLab alternative

– summer 2019 ??

Source code: <https://github.com/medvednikov/gitly>

8) **V blog** (<https://blog.vlang.io/>): Now it's powered by very basic blogging software, but in the future it will be a full featured light alternative to Discourse. It doesn’t use JavaScript!

**V forum**(<https://blog.vlang.io/forum>)

Source code: <https://github.com/medvednikov/vtalk>

9) **scc** (<https://github.com/boyter/scc/>): a tool for calculating LOC and so on, now supports V.

For a list of software built in V, consult <https://vlang.io/#software>

Real life examples (??)

Succes stories (??)

**1.2.10 Summary**

Here are the killer features of V:

* Emphasis on simplicity: *easy and quick to learn, learn it in 30 min*
* Syntactically lightweight: *easy to use*
* Fast compilation: *enhances productivity (such as the development cycle of a scripting language)*
* Fast executable code: *comparable to C*
* *Run everywhere* (through V’s compiling to C)
* Live code reloading: this should make possible self modifying code.
* Safer than Go
* Highly readable and self-documenting
* Static typing
* Consistent standard library
* Build *native graphical apps* that look native with the cross platform UI library: you no longer need to embed a browser to develop cross platform apps quickly.
* Painless deployment
* Easy dependency management
* Free and Open Source (MIT licensed)
* (??) Concurrency support: *write more efficient code by using multiple cores*