Five Years of Delve

20 Oct 2019

Alessandro Arzilli

First commit

- May 3, 2014, by Derek Parker

```
commit f6741acdde9a7d89cda8eebaac21cf40abecac90
Author: Derek Parker <parkerderek86@gmail.com>
Date:   Sat May 3 15:18:00 2014 -0500

    Add basic ignore config
```

Statistics

- 99 contributors
- Maintained primarily by Derek and me (I joined in June 2015)
- 7000 daily clones
- 1500 commits

Linux and Mac OS X support (2014)

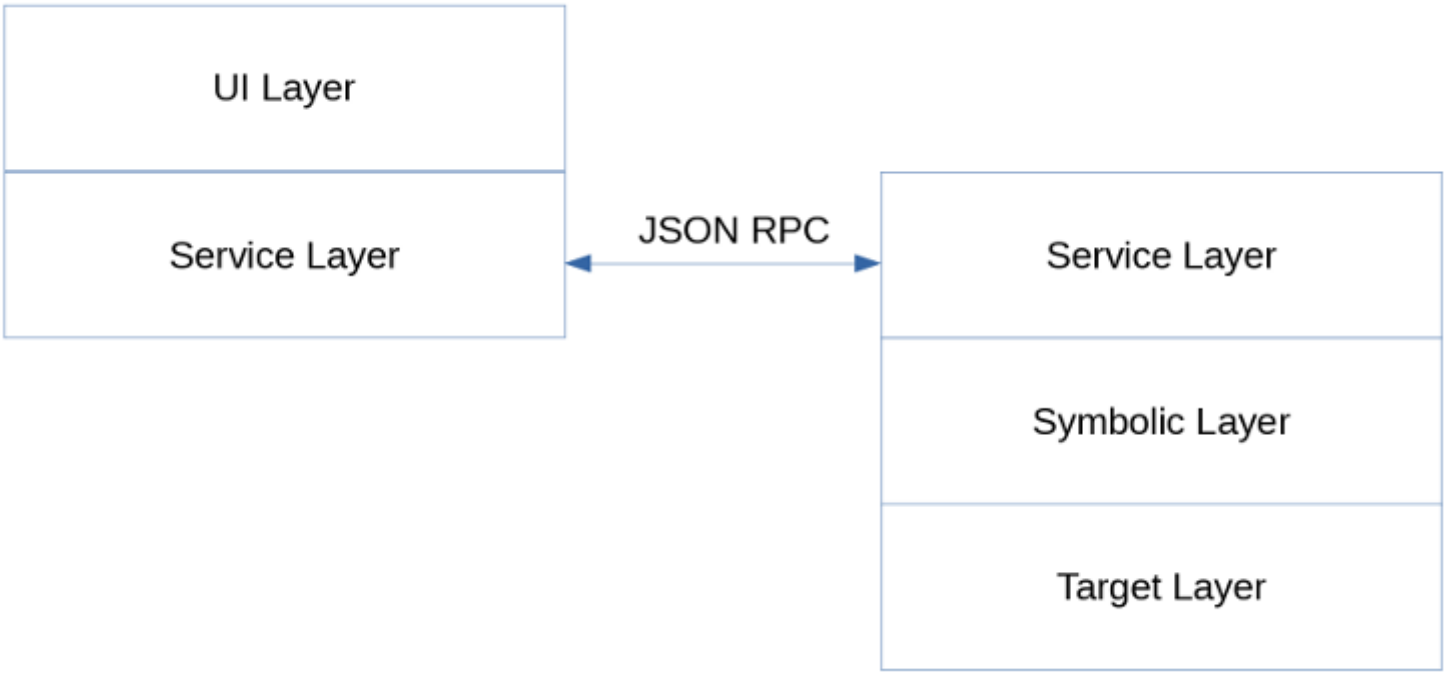
- Delve started its life on Linux
- OS X support was added very early
- Both were already there when I joined

```
commit 2d2d70641ec67593d09ee8264072fb0d11a11317
Author: Derek Parker <parkerderek86@gmail.com>
Date:   Tue Jan 13 20:37:10 2015 -0600

    (Mostly) working on OS X
```

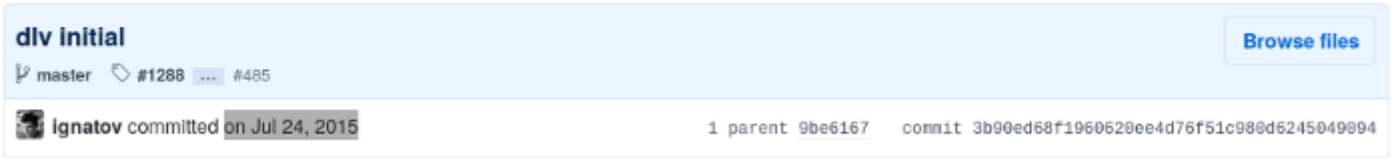
JSON-RPC

- Early feature of Delve (June 2015)
- Allows easy integration into IDEs/editors

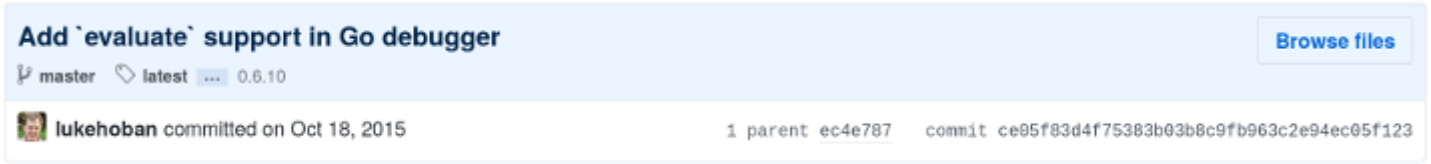


JSON-RPC integrations

- IntelliJ IDEA (now GoLand): July 2015



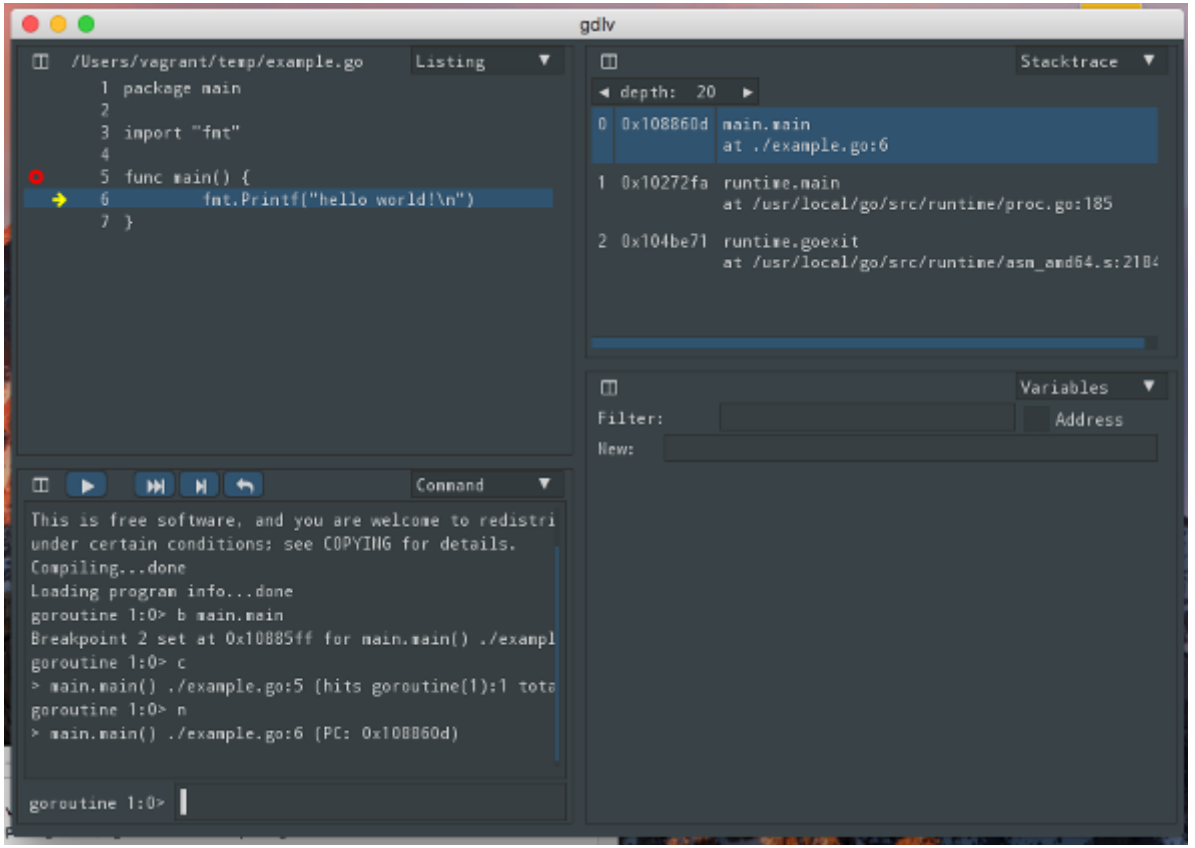
- VSCode (vscode-go): October 2015



- Atom, Sublime, Vim, Emacs...

JSON-RPC integrations

- Gdlv (GUI for Delve)



github.com/aarzilli/gdlv (https://github.com/aarzilli/gdlv)

Windows backend

- Landed in version 0.11.0-alpha, January 2016
- Pure Go

Mac OS X troubles

- Required cgo (Linux, and later Windows, impl. are pure Go)
- Kernel API needed by debuggers is completely undocumented
- Constantly dealing with bugs due to unspecified behavior
- OS X 10.12 broke our implementation to fix a security hole
- Debuggers on OS X have to be signed with a trusted certificate

Mac OS X troubles (2)

- Old install procedure on OS X:

```
* Open application “Keychain Access” (/Applications/Utilities/Keychain
  Access.app)
* Open menu /Keychain Access/Certificate Assistant/Create a Certificate...
* Choose a name (dlv-cert in the example), set “Identity Type” to “Self
  Signed Root”, set “Certificate Type” to “Code Signing” and select
  the “Let me override defaults”. Click “Continue”. You might want to
  extend the predefined 365 days period to 3650 days.
* Click several times on “Continue” until you get to the “Specify a
  Location For The Certificate” screen, then set “Keychain to System”.
* If you can't store the certificate in the “System” keychain, create
  it in the “login” keychain, then export it. You can then import it into
  the “System” keychain.
* In keychains select “System”, and you should find your new
  certificate. Use the context menu for the certificate, select “Get Info”,
  open the “Trust” item, and set “Code Signing” to “Always Trust”.
* [At least on Yosemite:] In keychains select category Keys -> dlv-cert ->
  right click -> GetInfo -> Access Control -> select "Allow all applications
  to access this item" -> Save Changes.
* You must quit “Keychain Access” application in order to use the
  certificate and restart “taskgated” service by killing the current
  running “taskgated” process. Alternatively you can restart your computer.
* Run the following: `GO15VENDOREXPERIMENT=1 CERT=dlv-cert make install`,
  which will install the binary and codesign it.
```

Mac OS X solutions / debugserver (May 2017)

- Debugserver is a small program developed along with LLDB
- It's distributed along with XCode by Apple
- It's a replacement for the same low-level debugger interface exposed by the (undocumented) Kernel API
- Implements the GDB Remote Serial Protocol
- The executable is signed by Apple

sourceware.org/gdb/onlinedocs/gdb/Remote-Protocol.html (https://sourceware.org/gdb/onlinedocs/gdb/Remote-Protocol.html)

github.com/llvm-mirror/lldb/blob/master/docs/lldb-gdb-remote.txt (https://github.com/llvm-mirror/lldb/blob/master/docs/lldb-gdb-remote.txt)

Mac OS X solutions / debugserver (May 2017) (2)

- New install procedure on OS X:

```
$ xcode-select --install
$ go get -u github.com/go-delve/delve/cmd/dlv
$ sudo /usr/sbin/DevToolsSecurity -enable
```

- (third line is optional)
- Landed on Delve 1.0.0-rc.1 in May 2017

Perks of supporting debugserver: Mozilla RR

- Mozilla RR also uses GDB Remote Serial Protocol
- RR is a "time travel" debugger backend
- It records the execution of a program and then you can play that recording forward or backwards (execution will be deterministic)
- By supporting debugserver we got Mozilla RR support almost for free

```
$ dlv --backend=rr debug ...
...
(dlv) rewind
...
```

Mozilla RR project (https://rr-project.org/)

Core file support

- Added for Linux core files in 1.0.0-rc.1 (February 2017)
- Added for Windows "minidump" files in 1.2.0 (October 2018)

Variable visibility (1)

- Not all improvements to debuggability come from improvements to Delve

```
func doSomethingInterface(in interface{}) error {
    switch n := in.(type) {
    case uint32:
        err := doSomethingUint32(n)
        if err != nil {
            return err
        }
    case *astruct:
        err := doSomethingAstruct(n)
        if err != nil {
            return err
        }
    default:
        return fmt.Errorf("unknown type %T", n)
    }
    return doSomethingFinish()
}
```

Variable visiblity (2)

- With Go 1.8 or earlier you'll get something like this:

```
(dlv) c
> main.doSomethingInterface() ./scope-example-full.go:18 (hits goroutine(1):1 total:1) (PC: 0x47d38b)
    14:                 return err
    15:             }
    16:         case *astruct:
    17:             err := doSomethingAstruct(n)
=>  18:             if err != nil {
    19:                 return err
    20:             }
(dlv) locals
n = 5230080
n#1 = (*main.astruct)(0xc42000e370)
n#2 = (unreadable invalid interface type: could not find str field)
err = error nil
err#4 = (unreadable invalid interface type: could not find str field)
```

- locals will show variables that aren't visible from the current line of code (and their value is mangled)

Variable visibility (3)

- Delve gets its info on functions, variables and types from a section of the executable called 'debug_info'
- The debug_info entry for doSomethingInterface with Go1.8 will look like this:

```
Subprogram "main.doSomethingInterface"
    Variable "n"     Location CFA-0xb0, Type uint32
    Variable "n#1"   Location CFA-0xa8, Type *main.astruct
    Variable "n#2"   Location CFA-0x98, Type interface {}
    Variable "err"   Location CFA-0x88, Type error
    Variable "err#4" Location CFA-0x78, Type error
    ...
```

- There is no way to tell which variables are scoped
- For the variables named 'n' you can guess based on their type
- No hope to guess the right 'err' variable

Variable visibility (4)

- Starting with Go1.9 the compiler started saving variable scopes in debug_info

```
Subprogram "main.doSomethingInterface"
    LexicalBlock, 0x48e755..0x48e79e
        Variable "n",   Location CFA-0xa0, Type uint32
        Variable "err", Location CFA-0x78, Type error
    LexicalBlock, 0x48e679..0x48e6c7
        Variable "n",   Location CFA-0x98, Type uint32
        Variable "err", Location CFA-0x68, Type error
    LexicalBlock, 0x48e7a3..0x48e891
        Variable "n",   Location CFA-0x88, Type interface {}
```

Variable visibility (5)

- Delve, starting with version 1.0.0-rc.2 (released in October 2017) can use this new information provided by Go 1.9:

```
> main.doSomethingInterface() ./scope-example-full.go:18 (PC: 0x4abfe0)
    15:             }
    16:         case *astruct:
    17:             err := doSomethingAstruct(n)
=>  18:             if err != nil {
    19:                 return err
    20:             }
    21:         default:
(dlv) locals
n = (*main.astruct)(0xc0000a2010)
err = error nil
```

Go 1.10 and later

- Constant entries in debug_info
- Improved instruction <-> line tables

Stable releases

- 1.0.0 release in February 2018
- Release cycle synchronized to Go
- Delve 1.0.0 supports Go 1.10
- Delve 1.1.0 supports Go 1.11
- Delve 1.2.0 supports Go 1.12...

Function call injection

- Needs runtime support due to GC/resizable stacks
- runtime.debugCallV1 added in Go 1.11
- Supported since Delve 1.1.0 (August 2018)
- Improved in 1.2.0, 1.3.0...

FreeBSD support

- New in Delve 1.3.0 (August 2019)
- Still experimental

PIE and plugin debugging support

- PIE support added in Delve 1.2.0
- Plugin support added in Delve 1.3.0
- All Linux only

Starlark scripting

- Added in 1.3.0
- Let's users add commands to delve
- Python-like language

Starlark scripting (2)

```
def command_goexcl(args):
    """Prints all goroutines not stopped in the function passed as argument."""
    excluded = 0
    start = 0
    while start >= 0:
        gr = goroutines(start, 10)
        start = gr.Nextg
        for g in gr.Goroutines:
            fn = g.UserCurrentLoc.Function
            if fn == None:
                print("Goroutine", g.ID, "User:", g.UserCurrentLoc.File, g.UserCurrentLoc.Line)
            elif fn.Name_ != args:
                print("Goroutine", g.ID, "User:", g.UserCurrentLoc.File, g.UserCurrentLoc.Line, fn.Name_
            else:
                excluded = excluded + 1
    print("Excluded", excluded, "goroutines")
```

Usage:

```
(dlv) source goexcl.star
(dlv) goexcl runtime.gopark
...
```

Future...

- Better cgo support
- Keep supporting newer versions of Go
- Better function call injection
- Support architectures other than amd64

Q&A

Recommended reading

Debugging
The Nine Indispensable Rules for Finding Even the Most Elusive Software and Hardware Problems
David J. Agans, 2002
American Management Association
ISBN 0−8144−7168−4

Thank you

20 Oct 2019

Alessandro Arzilli
alessandro.arzilli@gmail.com (mailto:alessandro.arzilli@gmail.com)