

Varlink Now!

All Systems Go! 2024
Berlin

Varlink? What's that?

- Extremely Simple IPC
- JSON formatted request over `SOCK_STREAM` socket, terminated by NUL byte
- Reply also JSON, terminated by NUL byte
- That's really all there is to it. KTHXBY

There's More

- Specs <https://varlink.org/>
- Bindings Available for Go, Rust, Python, Java,
...
- Been around since quite a while, initially with little adoption.

But... D-Bus!

- D-Bus is OK for many uses
- But it has (serious) limitations for others
- systemd deeply integrates with D-Bus, and that's not going to change any time soon

Problems with D-Bus

- Not available until late boot (not in initrd, for example)
- Too late for many (more than half of?) things in systemd
- systemd's way out so far: be our own little D-Bus daemon. Messy, complex, doesn't have proper D-Bus semantics, just uses serializations plus subset of header fields of messages, but otherwise is quite different (i.e. no matches, no name protocol, no peer-to-peer, ...)

Problems with D-Bus 2

- Cannot be used for many basic IPC services, since D-Bus itself consumes them, e.g. user/group database, logging, service lifecycle tracking, ... (otherwise cyclic loop: D-Bus uses IPC services implemented via D-Bus → deadlock)
- Cannot be used to stream data, i.e. acquiring list of things scalably. → no sensible flow control. Forces people into pattern that implies $n+1$ method calls for acquiring description of n objects. Racy.
- Slow: broker is bottleneck + roundtrip requires at least 4 context switches

Problems with D-Bus 3

- Security model is garbage, hence everyone uses polkit
- polkit hookup messy: implies at least 6 more process context switches
- Impossible to create & pin & subscribe to remote objects
- „Stuttering“ method calls

Problems with D-Bus 4

- There's so much more: type system has no nullable concept, not extensible without breaking API, redundant encoding, introspection doesn't know named structures, cryptic, XML, no documentation strings, property model pushes non-atomic behaviour, error system too naive, IPC connections not serializable/reloadable, not delegatable to containers/sandboxes without proxying/filtering, not really web friendly, no enums, global ordering requirement makes worker threads problematic..., etc. pp.

Problems with Varlink

- JSON is not 64bit integer clean (generally assumed to be 52bit integer clean)
- Format large integers as strings, and use null
- Marshalling a bit slower than D-Bus (~40%) for large objects, messages longer (but this is easily offset by reduction in roundtrips, and nicer strace'ability)
- However: Roundtrips kill you, slightly wasteful marshalling doesn't matter

Benefits of Varlink

- Connections pin objects, they entail both communication stream and stable reference
- No multiplexing, hence local ordering, not global ordering, which means compatible with thread pools
- Request/Response behaviour somewhat compatible with HTTP
- The simplicity of it and use of JSON makes it more digestible for web native folks, i.e. the rest of the world.

Benefits of Varlink 2

- Security model: no security of its own, relying on kernel (socket access control) + polkit.
- Delegation model: split your services into AF_UNIX socket inodes, mount them through into sandboxes/containers/...
- SO_PEERCREDS, SO_PEERSEC, SO_PEERPIDFD are your friends
- No „synthetic“ identifiers (no D-Bus object path or suchlike)

Benefits of Varlink 3

- Methods can have multiple replies („more“). Covers enumeration + subscription, streamable
- Compatible with per-connection socket activation: **very** easy to turn traditional Linux commands into IPC services. (This is a **killer** feature, do not underestimate.) No asynchronous, demuxing of multiple parallel calls on a single connection needed like on D-Bus.
- „Upgrade“ mechanism

Benefits of D-Bus

- Widely established on Linux
- Bindings for a myriad of languages
- Nice debugging tools such as D-Spy
- In some regards tighter type system, i.e. 64bit integers, signed/unsigned are a thing
- My take: on technical merits there's barely anything where D-Bus would have an edge. On adoption it (currently) wins, hands down.

Semantical Differences

- D-Bus method call selection: bus scope + service name + object path + interface + member name
- Varlink method call selection: entrypoint socket path + [service name + method name]

Semantical Differences 2

- No signals/matching as in D-Bus
- Instead: method call with „more“ flag
- Example where this matters:

```
varlinkctl call --more /run/systemd/io.systemd.Import io.systemd.Import.Pull  
'{"class":"machine","remote":"https://example.com/importme.tar.gz  
","local":"importme","type":"tar"}'
```

- Nota bene: We'll get notifications about progress and completion with one (!) method call, not with the signal matching mess that D-Bus would make from this.

Semantical Differences 3

- Introspection in D-Bus: formalized, audience not clear: probably not directly humans? Maybe indirectly humans? Language bindings?
- Introspection in Varlink: formalized, with clear audience: humans.

sd-varlink

- With systemd v257: sd-json + sd-varlink are part of libsystemd.so, just like sd-bus
- No documentation (yet)
- Example code: the systemd tree
- Don't wrap sd-json/sd-varlink for higher level languages, use native JSON libraries instead and add Varlink on top, it's easy.

varlinkctl

- Command line tool
- Counterpart to busctl

Omissions

- Varlink Spec defines various additional facets
systemd's Varlink logic (currently) does not bother with
- No service „Resolver“ discovery/registry, or „Bridge“ support
- No kernel integration
- No HTTP proxy

Additions

- File descriptor passing
- „pidfds are cool“ vs. „no side-channel“, in intractable opposition
- Our solution: allow us to do both
- In case of pidfd: boot id + pid + pidfd inode nr can be as good as pidfd.

Future

- D-Bus is not going to go away
- In systemd: newer APIs probably more Varlink, less D-Bus
- In systemd: existing APIs will slowly acquire Varlink interfaces
- Everyone else: Consider Varlink for **your** project, too!

The End