systemd's User Database API

Lennart Poettering
FOSDEM 2025
Brussels, Belgium

30 min

TL;DR

If your project provides user records to Linux systems, consider just implementing systemd's `io.systemd.UserDatabase` *Varlink IPC APIs for it.*

What is `io.systemd.UserDatabase`?

A simple Varlink IPC API

Knows three method calls:

- `GetUserRecord()`
- `GetGroupRecord()`
- `GetMemberships()`

Any service can implement this API to provide user/group records to the system.

Step Back: What is Varlink?

Trivial IPC system

Based on SOCK_STREAM sockets, with JSON marshalling.

Benefits: broker-less, works in early boot, simple, `strace`-able, rich type system, compatible with web world, sensible flow control, sensible security model, compatible with per-connection socket activation, and a lot more.

Used in systemd v242 and newer. 23 APIs defined by now. (compare: 14 D-Bus APIs)

systemd’s focus slowly moving from D-Bus to Varlink.

How does `io.systemd.UserDatabase` work?

In your service, simply bind AF_UNIX/SOCK_STREAM socket to `/run/systemd/userdb/<servicename>`.

Implement the aforementioned API.

🥳 Done. ✨

All provided records are now available in the system.

How does `io.systemd.UserDatabase` work? Part #2

A client which wants to resolve a user:

1. Iterates through all sockets in the `/run/systemd/userdb/` directory.
2. Connects to all of them, in parallel
3. Each time issuing the same GetUserRecord() method call
4. First positive reply is used, remaining queries terminated (except if enumeration is requested)

Unlike NSS there is no programmable algorithm which record to accept.

It's fast: everything in parallel.

Assumption is that user record collisions are resolved independently, and prior to the query.

How does `io.systemd.UserDatabase` work? Part #3

Alternatively: as a client just talk to the special `/run/systemd/userdb/io.systemd.Multiplexer` service.

Which does this all without iterating/concurrency, in a single simple method call.

What's a User Record?

`io.systemd.UserDatabase` operates with JSON user records.

Record format defined by systemd (https://systemd.io/USER_RECORD)

True superset of `struct passwd` + `struct spwd`

Also inspired by RFC 2307

Many additional fields: e.g. SSH keys, email address, location, icon name, blob references, disposition, last change timestamp, umask, environment, preferred language, additional language, nice level, resource limits, not before/not after timestamps, cgroup limits, auto-login, preferred session, stop delay, kill processes flag, pkcs11 auth fields, fido2 auth fields, recovery auth fields, self-modifiable fields controls, password hint, …

What’s a User Record, Part #2

User records are extensible, both in the specification and with project-specific fields.

No more sidecars! (Well…)

Per-machine sections

Privileged section (aka “shadow”)

Status section (with disk usage, last login, …)

… and more!

Closer Look at the API

Example:

```
$ varlinkctl introspect /run/systemd/userdb/io.systemd.Home
…
interface io.systemd.UserDatabase
…
method GetUserRecord(
        uid: ?int,
        userName: ?string,
        service: string
) -> (
        record: object,
        incomplete: ?bool
)
…
```

What You Get From This

- Instant glibc NSS (aka: `getpwnam()`) compatibility (both directions)
- Instant exposure of SSH public keys to SSH
- Nice value, environment variable, resource management, … enforcement by `pam_systemd` + `systemd-logind`
- Better control about the disposition of your user records (i.e. regular vs. system, …)
- Thus better control of default behaviour, regarding login behaviour, logging behaviour and so on.
- Non-C languages are considering acquiring user records via this IPC instead of wrapping glibc NSS directly
- `userdbctl` is a fun tool!
- GNOME integration

Implementation: `systemd-homed`

`io.systemd.UserDatabase` is implemented by `systemd-homed`

But is generic! Designed to be implemented by other user/group record providers, too.

New fields

If you need any new fields, and feel they are reasonably generic, let us know!

What You Get From This, Part #2

In other words:

Direct control how systemd deals with your users

Less maintenance work (no more NSS glue nor SSH glue to maintain)

Less stepping on each other’s toes

Your own fields

Nice tooling

Integration with systemd + other projects

Outlook

With v258: server side filtering by UID range/fuzzy name search/disposition search + aliases for user records

Later: allocation range queries

Out of focus: API to change user records

In focus: binding to Web accounts

The End