varlinkctl

- Description
- Commands
- Options
- Files & Directories
- Examples
- See Also

systemd

varlinkctl

1

varlinkctl

Introspect with and invoke Varlink services

varlinkctl

OPTIONS

info

ADDRESS

varlinkctl

OPTIONS

list-interfaces

ADDRESS

varlinkctl

OPTIONS

list-methods

ADDRESS

INTERFACE

varlinkctl

OPTIONS

introspect

ADDRESS

INTERFACE

varlinkctl

OPTIONS

call

ADDRESS

METHOD

ARGUMENTS

varlinkctl

OPTIONS

--exec

call

ADDRESS

METHOD

ARGUMENTS

--

CMDLINE

varlinkctl

OPTIONS

serve

METHOD

CMDLINE

varlinkctl

OPTIONS

list-registry

varlinkctl

OPTIONS

list-sockets

varlinkctl

OPTIONS

validate-idl

FILE

Description

`varlinkctl` may be used to introspect and invoke Varlink IPC services, following the UAPI.20 Varlink IPC specification.

Services are referenced by one of the following:

- A Varlink service reference starting with the `unix:` string, followed by an absolute `AF_UNIX` socket path, or by `@` and an arbitrary string (the latter for referencing sockets in the abstract namespace). In this case, a stream socket connection is made to the specified socket.
- A Varlink service reference starting with the `exec:` string, followed by an absolute path of a binary to execute. In this case, the specified process is forked off locally, with a connected stream socket passed in.
- A Varlink service reference starting with the `ssh-unix:` string, followed by an SSH host specification, followed by `:`, followed by an absolute `AF_UNIX` socket path. (This requires

OpenSSH 9.4 or newer on the server side, and abstract namespace sockets are not supported.)
- A Varlink service reference starting with the `ssh-exec:` string, followed by an SSH host specification, followed by `:`, followed by a command line. In this case, the command is invoked and the Varlink protocol is spoken on the standard input and output of the invoked command.

For convenience, these two simpler (redundant) service address syntaxes are also supported:

- A file system path to an `AF_UNIX` socket, either absolute (i.e. begins with `/`) or relative (in which case it must begin with `./`).
- A file system path to an executable, either absolute or relative (as above, must begin with `/` or `./`, respectively).

Commands

The following commands are understood:

`info` <ADDRESS>
Shows brief information about the specified service, including vendor name and list of implemented interfaces. Expects a service address in one of the formats described above.

`list-interfaces` <ADDRESS>
Shows a list of interfaces implemented by the specified service. Expects a service address in one of the formats described above.

`list-methods` <ADDRESS> [<INTERFACE…>]
Shows a list of methods implemented by the specified service. Expects a service address in one of the formats described above as well as one or more interface names. If no interface name is specified, lists all methods of all interfaces implemented by the service, otherwise just the methods in the specified interfaces.

If standard output and standard error are connected to a terminal, methods that are marked as experimental are highlighted and suffixed with `(experimental)`.

`introspect` <ADDRESS> [<INTERFACE…>]
Shows the interface definitions of the specified interfaces provided by the specified service. Expects a service address in one of the formats described above and optionally one or more Varlink interface names. If no interface names are specified, shows all provided interfaces by the service.

`call` <ADDRESS> <METHOD> [<ARGUMENTS>]
Calls the specified method of the specified service. Expects a service address in the format described above, a fully qualified Varlink method name, and a JSON arguments object. If the arguments object is not specified, it is read from STDIN instead. To pass an empty list of

parameters, specify the empty object `{}`.

The reply parameters are written as JSON objects to STDOUT.

If standard error is connected to a terminal and `--quiet` is not specified, the interface description is queried first, and a notice is shown if the method is marked as experimental.

`serve` <METHOD> <CMDLINE…>
Run a Varlink server that accepts protocol upgrade requests for the specified method and connects the upgraded connection to the standard input and output of the specified command. This can act as a server-side counterpart to `call` `--upgrade`.

The listening socket must be passed via socket activation (i.e. the `$LISTEN_FDS` protocol), making this command suitable for use in socket-activated service units. When a client calls the specified method with the upgrade flag, the server sends a reply confirming the upgrade, then forks and executes the given command line with the upgraded connection on its standard input and output.

This effectively turns any command that speaks a protocol over standard input/output into a Varlink service, discoverable via the service registry and authenticated via socket credentials. Because each connection is handled by a forked child process, the service unit can apply systemd's sandboxing options (such as `ProtectSystem=`, etc.) and does not operate in the caller's environment.

Use `--polkit-action=` to require polkit authorization of the client before the connection is upgraded.

`list-registry`
Shows a list of Varlink services currently registered in the service registry, plus their entrypoint sockets. (Currently, this simply enumerates the sockets and symlinked sockets in `/run/varlink/registry/`, see below.)

`list-sockets`
Shows a list of listening `AF_UNIX` stream sockets on the local system that are marked as Varlink entrypoints, along with whether the calling user has write (i.e. connect) access to each of them. A socket is considered a Varlink entrypoint if its inode carries the `user.varlink` extended attribute set to `entrypoint`.

Sockets are enumerated via the kernel's `sock_diag` netlink interface, and hence this is not restricted to sockets in `/run/varlink/registry/` (as `list-registry` is), but covers all file-system bound listening Varlink sockets, regardless of their location. This requires kernel support for extended attributes on socket inodes, which is available since Linux 7.0.

`validate-idl` [<FILE>]
Reads a Varlink interface definition file, parses and validates it, then outputs it with syntax highlighting. This checks for syntax and internal consistency of the interface. Expects a file name to read the interface definition from. If omitted, reads the interface definition from STDIN.

`help`
Shows command syntax help.

Options

The following options are understood:

`--more`
When used with `call`: expect multiple method replies. If this flag is set, the method call is sent with the `more` flag set, which tells the service to generate multiple replies, if needed. The command remains running until the service sends a reply message that indicates it is the last in the series (or if the configured timeout is reached, see below). This flag should be set only for method calls that support this mechanism.

If this mode is enabled, output is automatically switched to JSON-SEQ mode, so that individual reply objects can be easily discerned.

This switch has no effect on the method call timeout applied by default. Regardless of whether `--more` is specified or not, the default timeout will be 45s. Use `--timeout=` (see below) to change or disable the timeout. When invoking a method call that continuously returns updates, it is typically desirable to disable the timeout with `--timeout=infinity`. On the other hand, when invoking a `--more` method call for the purpose of enumerating objects (which likely will complete quickly), it is typically beneficial to leave the timeout logic enabled, for robustness reasons.

`-E`
A shortcut for `--more --timeout=infinity`. This switch is useful for method calls that implement subscription to a continuous stream of updates.

`--collect`
This is similar to `--more`, but collects all responses in a JSON array, and prints it, rather than in JSON-SEQ mode.

`--oneway`
When used with `call`: do not expect a method reply. If this flag is set, the method call is sent with the `oneway` flag set (the command exits immediately after), which tells the service not to generate a reply.

`--upgrade`
When used with `call`: request a protocol upgrade. The method call is sent with the `upgrade` flag set. The service is expected to send a single reply confirming the upgrade. After the reply, the Varlink protocol is no longer in effect on the connection.

If `--exec` is not specified, `varlinkctl` acts as a bidirectional proxy: data read from standard input is forwarded to the upgraded connection, and data received from the connection is written to

standard output.

If `--exec` is specified, the upgraded connection socket is placed on both standard input and standard output of the invoked process. This is similar to the regular `--exec` behavior (without `--upgrade`), which places the method call reply on standard input. The invoked process can thus simply read from and write to stdin/stdout to communicate over the upgraded protocol.

This option may not be combined with `--more`, `--oneway`, `--collect`, `--graceful=`, or `--push-fd=`.

`--json=MODE`
Selects the JSON output formatting, either `pretty` for nicely indented, colorized output, or `short` for terse output with minimal whitespace and no newlines. Defaults to `short`.

`-j`
Equivalent to `--json=pretty` when invoked interactively from a terminal. Otherwise, it is equivalent to `--json=short`, in particular when the output is piped to some other program.

`--quiet`; `-q`
Suppress output of method call replies.

`--graceful=`
Takes a qualified Varlink error name, i.e. an interface name, suffixed by an error name, separated by a dot, e.g. `org.varlink.service.InvalidParameter`. Ensures that, if a method call fails with the specified error, this will be treated as success, i.e. will cause the `varlinkctl` invocation to exit with a zero exit status. This option may be used more than once in order to treat multiple different errors as successes.

`--timeout=`
Expects a timeout in seconds as parameter. By default, a timeout of 45s is enforced. To turn off the timeout, specify `infinity` or an empty string.

`--exec`
Once the method call issued via `call` completed successfully, chainload the specified command line, with the method call output parameters serialized to JSON passed into standard input (and standard output and standard error inherited from the invoking process). Moreover any file descriptors passed back on the underlying communication socket are passed to the invoked process via the usual `$LISTEN_FDS` protocol. This functionality may be used to consume replies that come with associated file descriptors in a reasonable way.

Now that if `--exec` is specified the third parameter to `call` is not optional (i.e. the method call parameters).

`--push-fd=`
Takes a numeric file descriptor number as parameter. May be used to pass a file descriptor along with the method call, if the underlying transport supports this. May be used multiple times to pass multiple file descriptors, retaining the order in which they are specified. The specified file descriptors must be passed to the `varlinkctl` invocation. Optionally, in place of a numeric file

descriptor number an absolute or relative file system path (the latter must be prefixed with `./`) may be specified, which is opened in read-only mode.

`--polkit-action=ACTION`
When used with `serve`: before upgrading a connection, verify via polkit that the client is authorized for the specified polkit action ID. The client's identity is taken from the peer credentials of the connection. Clients running as root or as the same user as the server are always authorized. If authorization is denied, the method call fails with `org.varlink.service.PermissionDenied` and no command is executed. If polkit support is not available, all other clients are denied.

`--system`; `--user`
Determines whether to query to the per-system or per-user registry when using the `list-registry` command. By default, the per-system registry is queried.

Files & Directories

`/run/varlink/registry/`
Directory containing `AF_UNIX` entrypoint socket inodes (or symlinks to them) of well-known, public Varlink interfaces on the local system. They are named after the Varlink interface they implement.

Use `varlinkctl list-registry` to show the contents of this directory.

(Inodes that neither qualify as socket inodes nor as symlinks to them shall be ignored. A future extension might introduce regular files and directories to enhance the registry functionality.)

Examples

Investigating a Service

The following three commands inspect the `io.systemd.Resolve` service implemented by `systemd-resolved.service(8)`, listing general service information and implemented interfaces, and then displaying the interface definition of its primary interface:

```
$ varlinkctl info /run/systemd/resolve/io.systemd.Resolve
    Vendor: The systemd Project
   Product: systemd (systemd-resolved)
   Version: 254 (254-1522-g4790521^)
       URL: https://systemd.io/
Interfaces: io.systemd
```

```
            io.systemd.Resolve
            org.varlink.service
$ varlinkctl list-interfaces /run/systemd/resolve/io.systemd.Resolve
io.systemd
io.systemd.Resolve
org.varlink.service
$ varlinkctl introspect /run/systemd/resolve/io.systemd.Resolve io.systemd.Resolve
interface io.systemd.Resolve
type ResolvedAddress(
        ifindex: ?int,
        …
```

(Interface definition has been truncated in the example above, in the interest of brevity.)

Invoking a Method

The following command resolves a hostname via `systemd-resolved.service(8)`'s `ResolveHostname` method call.

```
$ varlinkctl call /run/systemd/resolve/io.systemd.Resolve io.systemd.Resolve.ResolveHostname
'{"name":"systemd.io","family":2}' -j
{
        "addresses" : [
                {
                        "ifindex" : 2,
                        "family" : 2,
                        "address" : [
                                185,
                                199,
                                111,
                                153
                        ]
                }
        ],
        "name" : "systemd.io",
        "flags" : 1048577
}
```

Investigating a Service Executable

The following command inspects the `/usr/lib/systemd/systemd-pcrextend` executable and the IPC APIs it provides. It then invokes a method on it:

```
# varlinkctl info /usr/lib/systemd/systemd-pcrextend
    Vendor: The systemd Project
   Product: systemd (systemd-pcrextend)
   Version: 254 (254-1536-g97734fb)
       URL: https://systemd.io/
Interfaces: io.systemd
            io.systemd.PCRExtend
            org.varlink.service
# varlinkctl introspect /usr/lib/systemd/systemd-pcrextend io.systemd.PCRExtend
interface io.systemd.PCRExtend

method Extend(
        pcr: int,
        text: ?string,
        data: ?string
) -> ()
# varlinkctl call /usr/lib/systemd/systemd-pcrextend io.systemd.PCRExtend.Extend
'{"pcr":15,"text":"foobar"}'
{}
```

Invoking a method remotely via SSH

The following command acquires a report about the identity of a remote host `somehost` from `systemd-hostnamed.service(8)` by connecting via SSH to the `AF_UNIX` socket the service listens on:

```
# varlinkctl call ssh-unix:somehost:/run/systemd/io.systemd.Hostname
io.systemd.Hostname.Describe '{}'
```

To invoke a Varlink service binary directly on the remote host, rather than talking to a service via `AF_UNIX` can be done like this:

```
# varlinkctl call ssh-exec:somehost:systemd-creds org.varlink.service.GetInfo '{}'
```

Serving a Sandboxed Decompressor via Protocol Upgrade

The following socket and service units expose `xz` decompression as a Varlink service. Clients connect and send compressed data over the upgraded connection, receiving decompressed output in return.

```
# /etc/systemd/system/varlink-decompress-xz.socket
[Socket]
ListenStream=/run/varlink/registry/com.example.Decompress.XZ

[Install]
WantedBy=sockets.target

# /etc/systemd/system/varlink-decompress-xz.service
[Service]
ExecStart=varlinkctl serve com.example.Decompress.XZ xz -d
DynamicUser=yes
PrivateNetwork=yes
ProtectSystem=strict
ProtectHome=yes
NoNewPrivileges=yes
SystemCallFilter=~@privileged @resources
MemoryMax=256M
```

A client can then decompress data through this service:

```
$ echo "hello" | xz | varlinkctl call --upgrade \
        unix:/run/varlink/registry/com.example.Decompress.XZ \
        com.example.Decompress.XZ '{}'
hello
```

For quick testing without unit files, `systemd-socket-activate` can be used to provide the listening socket:

```
$ systemd-socket-activate -l /tmp/decompress.sock -- varlinkctl serve
com.example.Decompress.XZ xz -d &
$ echo "hello" | xz | varlinkctl call --upgrade unix:/tmp/decompress.sock
com.example.Decompress.XZ '{}'
hello
```

See Also

`busctl(1)`, Varlink

□□□ #1
□ □□□ □□□ 2026-10-31 15:08:42 UTC

□ □□□ □□□ 2026-10-31 15:08:42 UTC