Rocky Enterprise Linux 9.2 Manual Pages on command 'dbus-broker.1'

$ man dbus-broker.1

DBUS-BROKER(1) User Commands DBUS-BROKER(1)

NAME

dbus-broker - D-Bus message broker

SYNOPSIS

dbus-broker [OPTIONS]

dbus-broker --version

dbus-broker --help

DESCRIPTION

dbus-broker is an implementation of the D-Bus Message Bus Specification [1]. Each instance provides a single, unique message bus that clients can connect to, and send messages over. The broker takes care of mes? sage mediation, access control, subscriptions, and bus control, accord? ing to the D-Bus specification.

dbus-broker is a pure implementation, meaning that it only implements the message mediation. It needs a controlling process that performs the bus setup and all external communication. dbus-broker-launch(1) is such a controller aiming at perfect compatibility to dbus-daemon(1), the D-Bus Reference Implementation. See dbus-broker-launch(1) for details

how to spawn a message bus.

This man-page documents the interface between dbus-broker and its con?

troller (e.g., dbus-broker-launch(1)).

OPTIONS

The following command-line options are supported. If an option is passed, which is not listed here, the broker will deny startup and exit with an error.

-h, --help

print usage information and exit immediately

--version

print build-version and exit immediately

--audit

enable logging to the linux audit subsystem (no-op if audit sup?

port was not compiled in; Default: off)

--controller=FD

use the inherited file-descriptor with the given number as the controlling socket (see CONTROLLER section; this option is mandatory)

--log FD

use the inherited file-descriptor with the given number to ac?

cess the system log (see LOGGING section; Default: no logging)

--machine-id=ID

set the machine-id to be advertised by the broker via the org.freedesktop.DBus interface (this option is mandatory and usually sourced from /etc/machine-id)

--max-bytes=BYTES

maximum number of bytes each user may allocate in the broker (Default: 16 MiB)

--max-fds=FDS

maximum number of file descriptors each user may allocate in the broker (Default: 64)

--max-matches=MATCHES

maximum number of match rules each user may allocate in the bro?

ker (Default: 16k)

--max-objects=OBJECTS

maximum total number of names, peers, pending replies, etc each user may allocate in the broker (Default: 16k)

CONTROLLER

Every instance of dbus-broker inherits a unix(7) socket from its parent process. This socket must be specified via the --controller option. The broker uses this socket to accept control commands from its parent process (or from whomever owns the other side of this socket, also called The Controller). This socket uses normal D-Bus P2P communica? tion. The interfaces provided on this socket are described in the API section.

By default, a broker instance is idle. That is, after forking and exe? cuting a broker, it starts with an empty list of bus-sockets to manage, as well as no way for clients to connect to it. The controller must use the controller interface to create listener sockets, specify the bus policy, create activatable names, and react to bus events.

The dbus-broker process never accesses any external resources other than those passed in either via the command-line or the controller in? terfaces. That is, no file-system access, no nss(5) calls, no external process communication, is performed by the broker. On the contrary, the broker never accesses any resources but the sockets provided to it by the controller. This is guaranteed by the implementation. At the same time, this implies that the controller is required to perform all ex? ternal resource acquisitions and communication on behalf of the broker (in case this is needed).

LOGGING

If a logging FD is provided via the --log command-line option, the bro? ker will log some information through this FD. Two different log-types are supported:

1. If the FD is a unix(7) SOCK_STREAM socket, information is logged as human-readable line-based chunks.
2. If the FD is a unix(7) SOCK_DGRAM socket, information is logged

as key/value based annotated data blocks. The format is compati? ble to the format used by the systemd-journal (though it is not dependent on systemd). This key/value based logging is a lot more verbose as the stream based logging. A lot of metadata is provided as separate keys, allowing precise tracing and interpre? tation of the logged data.

The broker has strict rules when it logs data. It logs during startup and shutdown, one message each to provide information on its setup and environment. At runtime, the broker only ever logs in unexpected situ? ations. That is, every message the broker logs at runtime was triggered by a malfunctioning client. If a system is properly set up, no runtime log-message will be triggered.

The situations where the broker logs are:

1. During startup and shutdown, the broker submits a short message including metadata about its controller, environment, and setup.
2. Whenever a client-request is denied by the policy, a message is logged including the affected client and policies.
3. Whenever a client exceeds its resource quota, a message is logged with information on the client.

API

The following interfaces are implemented by the broker on the respec? tive nodes. The controller is free to call these at any time. The con? troller connection is considered trusted. No resource accounting, nor access control is performed.

The controller itself is also required to implement interfaces to be used by the broker. See the section below for a list of interfaces on the controller.

```
node /org/bus1/DBus/Broker {
  interface org.bus1.DBus.Broker {
    # Create new activatable name @name, accounted on user @uid. The name
    # will be exposed by the controller as @path (which must fit the
    # template /org/bus1/DBus/Name/%).
    method AddName(o path, s name, u uid) -> ()
```

```
    # Add a listener socket to this bus. The listener socket must be
    # ready in listening mode and specified as @socket. As soon as this
    # call returns, incoming client connection attempts will be served
    # on this socket.
    # The listener is exposed by the controller as @path (which must fit
    # the template /org/bus1/DBus/Listener/%).
    # The policy for all clients connecting through this socket is
    # provided as @policy. See org.bus1.DBus.Listener.SetPolicy() for
    # details.
    method AddListener(o path, h socket, v policy) -> ()
    # This signal is raised according to client-requests of
    # org.freedesktop.DBus.UpdateActivationEnvironment().
    signal SetActivationEnvironment(a{ss} environment)
  }
}
node /org/bus1/DBus/Listener/% {
  interface org.bus1.DBus.Listener {
    # Release this listener. It will immediately be removed by the broker
    # and no more connections will be served on it. All clients connected
    # through this listener are forcefully disconnected.
    method Release() -> ()
    # Change the policy on this listener socket to @policy. The syntax of
    # the policy is still subject to change and not stable, yet.
    method SetPolicy(v policy) -> ()
  }
}
node /org/bus1/DBus/Name/% {
  interface org.bus1.DBus.Name {
    # Release this activatable name. It will be removed with immediate
    # effect by the broker. Note that the name is still valid to be
    # acquired by clients, though no activation-features will be
    # supported on this name.
    method Release() -> ()
```

```
    # Reset the activation state of this name. Any pending activation
    # requests are canceled. The call requires a serial number to be
    # passed along. This must be the serial number received by the last
    # activation even on this name. Calls for other serial numbers are
    # silently ignored and considered stale.
    method Reset(t serial) -> ()
    # This signal is sent whenever a client requests activation of this
    # name. Note that multiple activation requests are coalesced by the
    # broker. The controller can cancel outstanding requests via the
    # Reset() method.
    # The broker sends a serial number with the event. This number
    # represents the activation request and must be used when reacting
    # to the request with methods like Reset(). The serial number is
    # unique for each event, and is never reused. A serial number of 0
    # is never sent and considered invalid.
    signal Activate(t serial)
  }
}
```

The controller itself is required to implement the following interfaces on the given nodes. These interfaces are called by the broker to implement some parts of the driver-interface as defined by the D-Bus specification.

Note that all method-calls performed by the broker are always fully asynchronous. That is, regardless how long it takes to serve the request, the broker is still fully operational and might even send further requests to the controller.

A controller is free to implement these calls in a blocking fashion. However, it is up to the controller to make sure not to perform blocking recursive calls back into the broker (via any means).

```
node /org/bus1/DBus/Controller {
  interface org.bus1.DBus.Controller {
    # This function is called for each client-request of
    # org.freedesktop.DBus.ReloadConfig().
```

```
    method ReloadConfig() -> ()
  }
}
```

SEE ALSO

dbus-broker-launch(1) dbus-daemon(1)

NOTES

[1] D-Bus Specification:

https://dbus.freedesktop.org/doc/dbus-specification.html

DBUS-BROKER(1)