



**Full credit is given to the above companies including the Operating System (OS) that this PDF file was generated!**

### ***Rocky Enterprise Linux 9.2 Manual Pages on command 'org.freedesktop.machine1.5'***

***\$ man org.freedesktop.machine1.5***

ORG.FREEDESKTOP.MACHINE1(5)      org.freedesktop.machine1      ORG.FREEDESKTOP.MACHINE1(5)

NAME

org.freedesktop.machine1 - The D-Bus interface of systemd-machined

INTRODUCTION

systemd-machined.service(8) is a system service that keeps track of locally running virtual machines and containers. This page describes the D-Bus interface.

THE MANAGER OBJECT

The service exposes the following interfaces on the Manager object on the bus:

```
node /org/freedesktop/machine1 {
    interface org.freedesktop.machine1.Manager {
        methods:
            GetMachine(in s name,
                      out o machine);
            GetImage(in s name,
                   out o image);
            GetMachineByPID(in u pid,
                           out o machine);
            ListMachines(out a(ssso) machines);
            ListImages(out a(ssbttto) images);
            CreateMachine(in s name,
                        in ay id,
                        in s service,
                        in s class,
```

```
in u leader,  
in s root_directory,  
in a(sv) scope_properties,  
out o path);
```

```
CreateMachineWithNetwork(in s name,  
    in ay id,  
    in s service,  
    in s class,  
    in u leader,  
    in s root_directory,  
    in ai ifindices,  
    in a(sv) scope_properties,  
    out o path);
```

```
RegisterMachine(in s name,  
    in ay id,  
    in s service,  
    in s class,  
    in u leader,  
    in s root_directory,  
    out o path);
```

```
RegisterMachineWithNetwork(in s name,  
    in ay id,  
    in s service,  
    in s class,  
    in u leader,  
    in s root_directory,  
    in ai ifindices,  
    out o path);
```

```
UnregisterMachine(in s name);
```

```
TerminateMachine(in s id);
```

```
KillMachine(in s name,  
    in s who,  
    in i signal);
```

```
GetMachineAddresses(in s name,  
                    out a(iay) addresses);  
GetMachineOSRelease(in s name,  
                    out a{ss} fields);  
OpenMachinePTY(in s name,  
               out h pty,  
               out s pty_path);  
OpenMachineLogin(in s name,  
                 out h pty,  
                 out s pty_path);  
OpenMachineShell(in s name,  
                 in s user,  
                 in s path,  
                 in as args,  
                 in as environment,  
                 out h pty,  
                 out s pty_path);  
BindMountMachine(in s name,  
                 in s source,  
                 in s destination,  
                 in b read_only,  
                 in b mkdir);  
CopyFromMachine(in s name,  
                in s source,  
                in s destination);  
CopyToMachine(in s name,  
              in s source,  
              in s destination);  
OpenMachineRootDirectory(in s name,  
                          out h fd);  
GetMachineUIDShift(in s name,  
                   out u shift);  
RemoveImage(in s name);
```

```
RenamelImage(in s name,  
             in s new_name);  
ClonelImage(in s name,  
            in s new_name,  
            in b read_only);  
MarkImageReadOnly(in s name,  
                  in b read_only);  
GetImageHostname(in s name,  
                 out s hostname);  
GetImageMachineID(in s name,  
                  out ay id);  
GetImageMachineInfo(in s name,  
                    out a{ss} machine_info);  
GetImageOSRelease(in s name,  
                  out a{ss} os_release);  
SetPoolLimit(in t size);  
SetImageLimit(in s name,  
              in t size);  
CleanPool(in s mode,  
          out a(st) images);  
MapFromMachineUser(in s name,  
                   in u uid_inner,  
                   out u uid_outer);  
MapToMachineUser(in u uid_outer,  
                 out s machine_name,  
                 out o machine_path,  
                 out u uid_inner);  
MapFromMachineGroup(in s name,  
                    in u gid_inner,  
                    out u gid_outer);  
MapToMachineGroup(in u gid_outer,  
                  out s machine_name,  
                  out o machine_path,
```

```

        out u gid_inner);

signals:
    MachineNew(s machine,
               o path);
    MachineRemoved(s machine,
                  o path);

properties:
    @org.freedesktop.DBus.Property.EmitsChangedSignal("false")
    readonly s PoolPath = '...';
    @org.freedesktop.DBus.Property.EmitsChangedSignal("false")
    readonly t PoolUsage = ...;
    @org.freedesktop.DBus.Property.EmitsChangedSignal("false")
    readonly t PoolLimit = ...;
};

interface org.freedesktop.DBus.Peer { ... };
interface org.freedesktop.DBus.Introspectable { ... };
interface org.freedesktop.DBus.Properties { ... };
};

```

## Methods

GetMachine() may be used to get the machine object path for the machine with the specified name. Similarly, GetMachineByPID() gets the machine object the specified PID belongs to if there is any.

GetImage() may be used to get the image object path of the image with the specified name.

ListMachines() returns an array of all currently registered machines. The structures in the array consist of the following fields: machine name, machine class, an identifier for the service that registered the machine and the machine object path.

ListImages() returns an array of all currently known images. The structures in the array consist of the following fields: image name, type, read-only flag, creation time, modification time, current disk space, and image object path.

CreateMachine() may be used to register a new virtual machine or container with systemd-machined, creating a scope unit for it. It accepts the following arguments: a machine name chosen by the registrar, an optional UUID as a 32 byte array, a string that identifies the service that registers the machine, a class string, the PID of the leader

process of the machine, an optional root directory of the container, and an array of additional properties to use for the scope registration. The virtual machine name must be suitable as a hostname, and hence should follow the usual DNS hostname rules, as well as the Linux hostname restrictions. Specifically, only 7 bit ASCII is permitted, a maximum length of 64 characters is enforced, only characters from the set "a-zA-Z0-9-\_" are allowed, the name may not begin with a dot, and it may not contain two dots immediately following each other. Container and VM managers should ideally use the hostname used internally in the machine for this parameter. This recommendation is made in order to make the machine name naturally resolvable using `nss-myhostname(8)`. If a container manager needs to embed characters outside of the indicated range, escaping is required, possibly using "\_" as the escape character. Another (somewhat natural) option would be to utilize Internet IDNA encoding. The UUID is passed as a 32 byte array or, if no suitable UUID is available, an empty array (zero length) or zeroed out array shall be passed. The UUID should identify the virtual machine/container uniquely and should ideally be the same UUID that `/etc/machine-id` in the VM/container is initialized from. The service string can be free-form, but it is recommended to pass a short lowercase identifier like "systemd-nspawn", "libvirt-lxc" or similar. The class string should be either "container" or "vm" indicating whether the machine to register is of the respective class. The leader PID should be the host PID of the init process of the container or the encapsulating process of the VM. If the root directory of the container is known and available in the host's hierarchy, it should be passed. Otherwise, pass the empty string instead. Finally, the scope properties are passed as array in the same way as to `PID1's StartTransientUnit()` method. Calling this method will internally register a transient scope unit for the calling client (utilizing the passed `scope_properties`) and move the leader PID into it. The method returns an object path for the registered machine object that implements the `org.freedesktop.machine1.Machine` interface (see below). Also see the `New Control Group Interfaces[1]` for details about scope units and how to alter resource control settings on the created machine at runtime.

`RegisterMachine()` is similar to `CreateMachine()`. However, it only registers a machine and does not create a scope unit for it. Instead, the caller's unit is registered. We recommend to only use this method for container or VM managers that are run multiple times, one instance for each container/VM they manage, and are invoked as system services.

`CreateMachineWithNetwork()` and `RegisterMachineWithNetwork()` are similar to `CreateMachine()`

and RegisterMachine() but take an extra argument: an array of network interface indices that point towards the virtual machine or container. The interface indices should reference one or more network interfaces on the host that can be used to communicate with the guest. Commonly, the passed interface index refers to the host side of a "veth" link (in case of containers), a "tun"/"tap" link (in case of VMs), or the host side of a bridge interface that bridges access to the VM/container interfaces. Specifying this information is useful to enable support for link-local IPv6 communication to the machines since the scope field of sockaddr\_in6 can be initialized by the specified ifindex. nss-mymachines(8) makes use of this information.

KillMachine() sends a UNIX signal to the machine's processes. As its arguments, it takes a machine name (as originally passed to CreateMachine() or returned by ListMachines()), an identifier that specifies what precisely to send the signal to (either "leader" or "all"), and a numeric UNIX signal integer.

TerminateMachine() terminates a virtual machine, killing its processes. It takes a machine name as its only argument.

GetMachineAddresses() retrieves the IP addresses of a container. This method returns an array of pairs consisting of an address family specifier (AF\_INET or AF\_INET6) and a byte array containing the addresses. This is only supported for containers that make use of network namespacing.

GetMachineOSRelease() retrieves the OS release information of a container. This method returns an array of key value pairs read from the os-release(5) file in the container and is useful to identify the operating system used in a container.

OpenMachinePTY() allocates a pseudo TTY in the container and returns a file descriptor and its path. This is equivalent to transitioning into the container and invoking posix\_openpt(3).

OpenMachineLogin() allocates a pseudo TTY in the container and ensures that a getty login prompt of the container is running on the other end. It returns the file descriptor of the PTY and the PTY path. This is useful for acquiring a pty with a login prompt from the container.

OpenMachineShell() allocates a pseudo TTY in the container, as the specified user, and invokes the executable at the specified path with a list of arguments (starting from argv[0]) and an environment block. It then returns the file descriptor of the PTY and the PTY path.

`BindMountMachine()` bind mounts a file or directory from the host into the container. Its arguments consist of a machine name, the source directory on the host, the destination directory in the container, and two booleans, one indicating whether the bind mount shall be read-only, the other indicating whether the destination mount point shall be created first, if it is missing.

`CopyFromMachine()` copies files or directories from a container into the host. It takes a container name, a source directory in the container and a destination directory on the host as arguments. `CopyToMachine()` does the opposite and copies files from a source directory on the host into a destination directory in the container.

`RemoveImage()` removes the image with the specified name.

`RenameImage()` renames the specified image.

`CloneImage()` clones the specified image under a new name. It also takes a boolean argument indicating whether the resulting image shall be read-only or not.

`MarkImageReadOnly()` toggles the read-only flag of an image.

`SetPoolLimit()` sets an overall quota limit on the pool of images.

`SetImageLimit()` sets a per-image quota limit.

`MapFromMachineUser()`, `MapToMachineUser()`, `MapFromMachineGroup()`, and `MapToMachineGroup()` may be used to map UIDs/GIDs from the host user namespace to a container user namespace or vice versa.

## Signals

`MachineNew` and `MachineRemoved` are sent whenever a new machine is registered or removed.

These signals carry the machine name and the object path to the corresponding `org.freedesktop.machine1.Machine` interface (see below).

## Properties

`PoolPath` specifies the file system path where images are written to.

`PoolUsage` specifies the current usage size of the image pool in bytes.

`PoolLimit` specifies the size limit of the image pool in bytes.

## MACHINE OBJECTS

```
node /org/freedesktop/machine1/machine/rawhide {
    interface org.freedesktop.machine1.Machine {
        methods:
            Terminate();
            Kill(in s who,
```



```

    in i signal);
GetAddresses(out a(iay) addresses);
GetOSRelease(out a{ss} fields);
GetUIDShift(out u shift);
OpenPTY(out h pty,
        out s pty_path);
OpenLogin(out h pty,
        out s pty_path);
OpenShell(in s user,
        in s path,
        in as args,
        in as environment,
        out h pty,
        out s pty_path);
BindMount(in s source,
        in s destination,
        in b read_only,
        in b mkdir);
CopyFrom(in s source,
        in s destination);
CopyTo(in s source,
        in s destination);
OpenRootDirectory(out h fd);
properties:
    @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
    readonly s Name = '...';
    @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
    readonly ay Id = [...];
    @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
    readonly t Timestamp = ...;
    @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
    readonly t TimestampMonotonic = ...;
    @org.freedesktop.DBus.Property.EmitsChangedSignal("const")

```

```

readonly s Service = '...';
@org.freedesktop.DBus.Property.EmitsChangedSignal("const")
readonly s Unit = '...';
@org.freedesktop.DBus.Property.EmitsChangedSignal("const")
readonly u Leader = ...;
@org.freedesktop.DBus.Property.EmitsChangedSignal("const")
readonly s Class = '...';
@org.freedesktop.DBus.Property.EmitsChangedSignal("const")
readonly s RootDirectory = '...';
@org.freedesktop.DBus.Property.EmitsChangedSignal("const")
readonly ai NetworkInterfaces = [...];
@org.freedesktop.DBus.Property.EmitsChangedSignal("false")
readonly s State = '...';
};
interface org.freedesktop.DBus.Peer { ... };
interface org.freedesktop.DBus.Introspectable { ... };
interface org.freedesktop.DBus.Properties { ... };
};

```

## Methods

Terminate() and Kill() terminate/kill the machine. These methods take the same arguments as TerminateMachine() and KillMachine() on the Manager interface, respectively.

GetAddresses() and GetOSRelease() get the IP address and OS release information from the machine. These methods take the same arguments as GetMachineAddresses() and GetMachineOSRelease() of the Manager interface, respectively.

## Properties

Name is the machine name as it was passed in during registration with CreateMachine() on the manager object.

Id is the machine UUID.

Timestamp and TimestampMonotonic are the realtime and monotonic timestamps when the virtual machines were created in microseconds since the epoch.

Service contains a short string identifying the registering service as passed in during registration of the machine.

Unit is the systemd scope or service unit name for the machine.

Leader is the PID of the leader process of the machine.

Class is the class of the machine and is either the string "vm" (for real VMs based on virtualized hardware) or "container" (for light-weight userspace virtualization sharing the same kernel as the host).

RootDirectory is the root directory of the container if it is known and applicable or the empty string.

NetworkInterfaces contains an array of network interface indices that point towards the container, the VM or the host. For details about this information see the description of `CreateMachineWithNetwork()` above.

State is the state of the machine and is one of "opening", "running", or "closing". Note that the state machine is not considered part of the API and states might be removed or added without this being considered API breakage.

## EXAMPLES

Example 1. Introspect `org.freedesktop.machine1.Manager` on the bus

```
$ gdbus introspect --system \  
  --dest org.freedesktop.machine1 \  
  --object-path /org/freedesktop/machine1
```

Example 2. Introspect `org.freedesktop.machine1.Machine` on the bus

```
$ gdbus introspect --system \  
  --dest org.freedesktop.machine1 \  
  --object-path /org/freedesktop/machine1/machine/rawhide
```

## VERSIONING

These D-Bus interfaces follow the usual interface versioning guidelines[2].

## NOTES

### 1. New Control Group Interfaces

<https://www.freedesktop.org/wiki/Software/systemd/ControlGroupInterface/>

### 2. the usual interface versioning guidelines

<http://0pointer.de/blog/projects/versioning-dbus.html>