NAME

org.freedesktop.login1 − The D−Bus interface of systemd−logind

INTRODUCTION

systemd-logind.service(8) is a system service that keeps track of user logins and seats.

The daemon provides both a C library interface as well as a D−Bus interface. The library interface may be used to introspect and watch the state of user logins and seats. The bus interface provides the same functionality but in addition may also be used to make changes to the system state. For more information please consult **sd-login**(3).

THE MANAGER OBJECT

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

```
node /org/freedesktop/login1 {
  interface org.freedesktop.login1.Manager {
    methods:
      GetSession(in  s session_id,
                 out o object_path);
      GetSessionByPID(in  u pid,
                      out o object_path);
      GetUser(in  u uid,
              out o object_path);
      GetUserByPID(in  u pid,
                   out o object_path);
      GetSeat(in  s seat_id,
              out o object_path);
      ListSessions(out a(susso) sessions);
      ListUsers(out a(uso) users);
      ListSeats(out a(so) seats);
      ListInhibitors(out a(ssssuu) inhibitors);
      CreateSession(in  u uid,
                    in  u pid,
                    in  s service,
                    in  s type,
                    in  s class,
                    in  s desktop,
                    in  s seat_id,
                    in  u vtnr,
                    in  s tty,
                    in  s display,
                    in  b remote,
                    in  s remote_user,
                    in  s remote_host,
                    in  a(sv) properties,
                    out s session_id,
                    out o object_path,
                    out s runtime_path,
                    out h fifo_fd,
                    out u uid,
                    out s seat_id,
                    out u vtnr,
                    out b existing);
      ReleaseSession(in  s session_id);
      ActivateSession(in  s session_id);
      ActivateSessionOnSeat(in  s session_id,
```

```
                    in  s seat_id);
LockSession(in  s session_id);
UnlockSession(in  s session_id);
LockSessions();
UnlockSessions();
KillSession(in  s session_id,
            in  s who,
            in  i signal_number);
KillUser(in  u uid,
         in  i signal_number);
TerminateSession(in  s session_id);
TerminateUser(in  u uid);
TerminateSeat(in  s seat_id);
SetUserLinger(in  u uid,
              in  b enable,
              in  b interactive);
AttachDevice(in  s seat_id,
             in  s sysfs_path,
             in  b interactive);
FlushDevices(in  b interactive);
PowerOff(in  b interactive);
PowerOffWithFlags(in  t flags);
Reboot(in  b interactive);
RebootWithFlags(in  t flags);
Halt(in  b interactive);
HaltWithFlags(in  t flags);
Suspend(in  b interactive);
SuspendWithFlags(in  t flags);
Hibernate(in  b interactive);
HibernateWithFlags(in  t flags);
HybridSleep(in  b interactive);
HybridSleepWithFlags(in  t flags);
SuspendThenHibernate(in  b interactive);
SuspendThenHibernateWithFlags(in  t flags);
CanPowerOff(out s result);
CanReboot(out s result);
CanHalt(out s result);
CanSuspend(out s result);
CanHibernate(out s result);
CanHybridSleep(out s result);
CanSuspendThenHibernate(out s result);
ScheduleShutdown(in  s type,
                 in  t usec);
CancelScheduledShutdown(out b cancelled);
Inhibit(in  s what,
        in  s who,
        in  s why,
        in  s mode,
        out h pipe_fd);
CanRebootParameter(out s result);
SetRebootParameter(in  s parameter);
CanRebootToFirmwareSetup(out s result);
SetRebootToFirmwareSetup(in  b enable);
CanRebootToBootLoaderMenu(out s result);
```

```
  SetRebootToBootLoaderMenu(in  t timeout);
  CanRebootToBootLoaderEntry(out s result);
  SetRebootToBootLoaderEntry(in  s boot_loader_entry);
  SetWallMessage(in  s wall_message,
                 in  b enable);
signals:
  SessionNew(s session_id,
             o object_path);
  SessionRemoved(s session_id,
                 o object_path);
  UserNew(u uid,
          o object_path);
  UserRemoved(u uid,
              o object_path);
  SeatNew(s seat_id,
          o object_path);
  SeatRemoved(s seat_id,
              o object_path);
  PrepareForShutdown(b start);
  PrepareForSleep(b start);
properties:
  @org.freedesktop.DBus.Property.EmitsChangedSignal("false")
  @org.freedesktop.systemd1.Privileged("true")
  readwrite b EnableWallMessages = ...;
  @org.freedesktop.DBus.Property.EmitsChangedSignal("false")
  @org.freedesktop.systemd1.Privileged("true")
  readwrite s WallMessage = '...';
  @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
  readonly u NAutoVTs = ...;
  @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
  readonly as KillOnlyUsers = ['...', ...];
  @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
  readonly as KillExcludeUsers = ['...', ...];
  @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
  readonly b KillUserProcesses = ...;
  @org.freedesktop.DBus.Property.EmitsChangedSignal("false")
  readonly s RebootParameter = '...';
  @org.freedesktop.DBus.Property.EmitsChangedSignal("false")
  readonly b RebootToFirmwareSetup = ...;
  @org.freedesktop.DBus.Property.EmitsChangedSignal("false")
  readonly t RebootToBootLoaderMenu = ...;
  @org.freedesktop.DBus.Property.EmitsChangedSignal("false")
  readonly s RebootToBootLoaderEntry = '...';
  @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
  readonly as BootLoaderEntries = ['...', ...];
  readonly b IdleHint = ...;
  readonly t IdleSinceHint = ...;
  readonly t IdleSinceHintMonotonic = ...;
  readonly s BlockInhibited = '...';
  readonly s DelayInhibited = '...';
  @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
  readonly t InhibitDelayMaxUSec = ...;
  @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
  readonly t UserStopDelayUSec = ...;
```

```
        @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
        readonly s HandlePowerKey = '...';
        @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
        readonly s HandleSuspendKey = '...';
        @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
        readonly s HandleHibernateKey = '...';
        @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
        readonly s HandleLidSwitch = '...';
        @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
        readonly s HandleLidSwitchExternalPower = '...';
        @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
        readonly s HandleLidSwitchDocked = '...';
        @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
        readonly t HoldoffTimeoutUSec = ...;
        @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
        readonly s IdleAction = '...';
        @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
        readonly t IdleActionUSec = ...;
        @org.freedesktop.DBus.Property.EmitsChangedSignal("false")
        readonly b PreparingForShutdown = ...;
        @org.freedesktop.DBus.Property.EmitsChangedSignal("false")
        readonly b PreparingForSleep = ...;
        @org.freedesktop.DBus.Property.EmitsChangedSignal("false")
        readonly (st) ScheduledShutdown = ...;
        @org.freedesktop.DBus.Property.EmitsChangedSignal("false")
        readonly b Docked = ...;
        @org.freedesktop.DBus.Property.EmitsChangedSignal("false")
        readonly b LidClosed = ...;
        @org.freedesktop.DBus.Property.EmitsChangedSignal("false")
        readonly b OnExternalPower = ...;
        @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
        readonly b RemoveIPC = ...;
        @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
        readonly t RuntimeDirectorySize = ...;
        @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
        readonly t RuntimeDirectoryInodesMax = ...;
        @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
        readonly t InhibitorsMax = ...;
        @org.freedesktop.DBus.Property.EmitsChangedSignal("false")
        readonly t NCurrentInhibitors = ...;
        @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
        readonly t SessionsMax = ...;
        @org.freedesktop.DBus.Property.EmitsChangedSignal("false")
        readonly t NCurrentSessions = ...;
    };
    interface org.freedesktop.DBus.Peer { ... };
    interface org.freedesktop.DBus.Introspectable { ... };
    interface org.freedesktop.DBus.Properties { ... };
};
```

Methods

GetSession() may be used to get the session object path for the session with the specified ID. Similarly, **GetUser()** and **GetSeat()** get the user and seat objects, respectively. **GetSessionByPID()** and **GetUserByPID()** get the session/user object the specified PID belongs to if there is any.

ListSessions() returns an array of all current sessions. The structures in the array consist of the following fields: session id, user id, user name, seat id, session object path. If a session does not have a seat attached, the seat id field will be an empty string.

ListUsers() returns an array of all currently logged in users. The structures in the array consist of the following fields: user id, user name, user object path.

ListSeats() returns an array of all currently available seats. The structure in the array consists of the following fields: seat id, seat object path.

ListInhibitors() lists all currently active inhibitors. It returns an array of structures consisting of *what*, *who*, *why*, *mode*, *uid* (user ID), and *pid* (process ID).

CreateSession() and **ReleaseSession()** may be used to open or close login sessions. These calls should *never* be invoked directly by clients. Creating/closing sessions is exclusively the job of PAM and its **pam_systemd**(8) module.

ActivateSession() brings the session with the specified ID into the foreground. **ActivateSessionOnSeat()** does the same, but only if the seat id matches.

LockSession() asks the session with the specified ID to activate the screen lock. **UnlockSession()** asks the session with the specified ID to remove an active screen lock, if there is any. This is implemented by sending out the Lock() and Unlock() signals from the respective session object which session managers are supposed to listen on.

LockSessions() asks all sessions to activate their screen locks. This may be used to lock access to the entire machine in one action. Similarly, **UnlockSessions()** asks all sessions to deactivate their screen locks.

KillSession() may be used to send a Unix signal to one or all processes of a session. As arguments it takes the session id, either the string "leader" or "all" and a signal number. If "leader" is passed only the session "leader" is killed. If "all" is passed all processes of the session are killed.

KillUser() may be used to send a Unix signal to all processes of a user. As arguments it takes the user id and a signal number.

TerminateSession(), **TerminateUser()**, **TerminateSeat()** may be used to forcibly terminate one specific session, all processes of a user, and all sessions attached to a specific seat, respectively. The session, user, and seat are identified by their respective IDs.

SetUserLinger() enables or disables user lingering. If enabled, the runtime directory of a user is kept around and they may continue to run processes while logged out. If disabled, the runtime directory goes away as soon as they log out. **SetUserLinger()** expects three arguments: the UID, a boolean whether to enable/disable and a boolean controlling the **polkit**[1] authorization interactivity (see below). Note that the user linger state is persistently stored on disk.

AttachDevice() may be used to assign a specific device to a specific seat. The device is identified by its /sys/ path and must be eligible for seat assignments. **AttachDevice()** takes three arguments: the seat id, the sysfs path, and a boolean for controlling polkit interactivity (see below). Device assignments are persistently stored on disk. To create a new seat, simply specify a previously unused seat id. For more information about the seat assignment logic see **sd-login**(3).

FlushDevices() removes all explicit seat assignments for devices, resetting all assignments to the automatic defaults. The only argument it takes is the polkit interactivity boolean (see below).

PowerOff(), **Reboot()**, **Halt()**, **Suspend()**, and **Hibernate()** result in the system being powered off, rebooted, halted (shut down without turning off power), suspended (the system state is saved to RAM and the CPU is turned off), or hibernated (the system state is saved to disk and the machine is powered down). **HybridSleep()** results in the system entering a hybrid−sleep mode, i.e. the system is both hibernated and

suspended. **SuspendThenHibernate()** results in the system being suspended, then later woken using an RTC timer and hibernated. The only argument is the polkit interactivity boolean *interactive* (see below). The main purpose of these calls is that they enforce polkit policy and hence allow powering off/rebooting/suspending/hibernating even by unprivileged users. They also enforce inhibition locks for non−privileged users. UIs should expose these calls as the primary mechanism to poweroff/reboot/suspend/hibernate the machine. Methods **PowerOffWithFlags()**, **RebootWithFlags()**, **HaltWithFlags()**, **SuspendWithFlags()**, **HibernateWithFlags()**, **HybridSleepWithFlags()** and **SuspendThenHibernateWithFlags()** add *flags* to allow for extendability, defined as follows:

```
#define SD_LOGIND_ROOT_CHECK_INHIBITORS  (UINT64_C(1) << 0)
#define SD_LOGIND_KEXEC_REBOOT           (UINT64_C(1) << 1)
```

When the *flags* is 0 then these methods behave just like the versions without flags. When **SD_LOGIND_ROOT_CHECK_INHIBITORS** (0x01) is set, active inhibitors are honoured for privileged users too. When **SD_LOGIND_KEXEC_REBOOT** (0x02) is set, then **RebootWithFlags()** perform kexec reboot if kexec kernel is loaded.

SetRebootParameter() sets a parameter for a subsequent reboot operation. See the description of **reboot** in **systemctl**(1) and **reboot**(2) for more information.

SetRebootToFirmwareSetup(), **SetRebootToBootLoaderMenu()**, and **SetRebootToBootLoaderEntry()** configure the action to be taken from the boot loader after a reboot: respectively entering firmware setup mode, the boot loader menu, or a specific boot loader entry. See **systemctl**(1) for the corresponding command line interface.

CanPowerOff(), **CanReboot()**, **CanHalt()**, **CanSuspend()**, **CanHibernate()**, **CanHybridSleep()**, **CanSuspendThenHibernate()**, **CanRebootParameter()**, **CanRebootToFirmwareSetup()**, **CanRebootToBootLoaderMenu()**, and **CanRebootToBootLoaderEntry()** test whether the system supports the respective operation and whether the calling user is allowed to execute it. Returns one of "na", "yes", "no", and "challenge". If "na" is returned, the operation is not available because hardware, kernel, or drivers do not support it. If "yes" is returned, the operation is supported and the user may execute the operation without further authentication. If "no" is returned, the operation is available but the user is not allowed to execute the operation. If "challenge" is returned, the operation is available but only after authorization.

ScheduleShutdown() schedules a shutdown operation *type* at time *usec* in microseconds since the UNIX epoch. *type* can be one of "poweroff", "dry−poweroff", "reboot", "dry−reboot", "halt", and "dry−halt". (The "dry−" variants do not actually execute the shutdown action.) **CancelScheduledShutdown()** cancels a scheduled shutdown. The output parameter *cancelled* is true if a shutdown operation was scheduled.

SetWallMessage() sets the wall message (the message that will be sent out to all terminals and stored in a **utmp**(5) record) for a subsequent scheduled shutdown operation. The parameter *wall_message* specifies the shutdown reason (and may be empty) which will be included in the shutdown message. The parameter *enable* specifies whether to print a wall message on shutdown.

Inhibit() creates an inhibition lock. It takes four parameters: *what*, *who*, *why*, and *mode*. *what* is one or more of "shutdown", "sleep", "idle", "handle−power−key", "handle−suspend−key", "handle−hibernate−key", "handle−lid−switch", separated by colons, for inhibiting poweroff/reboot, suspend/hibernate, the automatic idle logic, or hardware key handling. *who* should be a short human readable string identifying the application taking the lock. *why* should be a short human readable string identifying the reason why the lock is taken. Finally, *mode* is either "block" or "delay" which encodes whether the inhibit shall be consider mandatory or whether it should just delay the operation to a certain maximum time. The method returns a file descriptor. The lock is released the moment this file descriptor and all its duplicates are closed. For more information on the inhibition logic see **Inhibitor Locks**[2].

Signals

Whenever the inhibition state or idle hint changes, **PropertyChanged** signals are sent out to which clients can subscribe.

The **SessionNew**, **SessionRemoved**, **UserNew**, **UserRemoved**, **SeatNew**, and **SeatRemoved** signals are sent each time a session is created or removed, a user logs in or out, or a seat is added or removed. They each contain the ID of the object plus the object path.

The **PrepareForShutdown()** and **PrepareForSleep()** signals are sent right before (with the argument "true") or after (with the argument "false") the system goes down for reboot/poweroff and suspend/hibernate, respectively. This may be used by applications to save data on disk, release memory, or do other jobs that should be done shortly before shutdown/sleep, in conjunction with delay inhibitor locks. After completion of this work they should release their inhibition locks in order to not delay the operation any further. For more information see **Inhibitor Locks**[2].

Properties

Most properties simply reflect the configuration, see **logind.conf**(5). This includes: *NAutoVTs*, *KillOnlyUsers*, *KillExcludeUsers*, *KillUserProcesses*, *IdleAction*, *InhibitDelayMaxUSec*, *InhibitorsMax*, *UserStopDelayUSec*, *HandlePowerKey*, *HandleSuspendKey*, *HandleHibernateKey*, *HandleLidSwitch*, *HandleLidSwitchExternalPower*, *HandleLidSwitchDocked*, *IdleActionUSec*, *HoldoffTimeoutUSec*, *RemoveIPC*, *RuntimeDirectorySize*, *RuntimeDirectoryInodesMax*, *InhibitorsMax*, and *SessionsMax*.

The *IdleHint* property reflects the idle hint state of the system. If the system is idle it might get into automatic suspend or shutdown depending on the configuration.

IdleSinceHint and *IdleSinceHintMonotonic* encode the timestamps of the last change of the idle hint boolean, in **CLOCK_REALTIME** and **CLOCK_MONOTONIC** timestamps, respectively, in microseconds since the epoch.

The *BlockInhibited* and *DelayInhibited* properties encode the currently active locks of the respective modes. They are colon separated lists of "shutdown", "sleep", and "idle" (see above).

NCurrentSessions and *NCurrentInhibitors* contain the number of currently registered sessions and inhibitors.

The *BootLoaderEntries* property contains a list of boot loader entries. This includes boot loader entries defined in configuration and any additional loader entries reported by the boot loader. See **systemd-boot**(7) for more information.

The *PreparingForShutdown* and *PreparingForSleep* boolean properties are true during the interval between the two **PrepareForShutdown** and **PrepareForSleep** signals respectively. Note that these properties do not send out **PropertyChanged** signals.

The *RebootParameter* property shows the value set with the **SetRebootParameter()** method described above.

ScheduledShutdown shows the value pair set with the **ScheduleShutdown()** method described above.

RebootToFirmwareSetup, *RebootToBootLoaderMenu*, and *RebootToBootLoaderEntry* are true when the respective post−reboot operation was selected with **SetRebootToFirmwareSetup**, **SetRebootToBootLoaderMenu**, or **SetRebootToBootLoaderEntry**.

The *WallMessage* and *EnableWallMessages* properties reflect the shutdown reason and wall message enablement switch which can be set with the **SetWallMessage()** method described above.

Docked is true if the machine is connected to a dock. *LidClosed* is true when the lid (of a laptop) is closed. *OnExternalPower* is true when the machine is connected to an external power supply.

Security

A number of operations are protected via the polkit privilege system. **SetUserLinger()** requires the org.freedesktop.login1.set−user−linger privilege. **AttachDevice()** requires org.freedesktop.login1.attach−device and **FlushDevices()** requires org.freedesktop.login1.flush−devices. **PowerOff()**, **Reboot()**, **Halt()**, **Suspend()**, **Hibernate()** require org.freedesktop.login1.power−off, org.freedesktop.login1.power−off−multiple−sessions, org.freedesktop.login1.power−off−ignore−inhibit, org.freedesktop.login1.reboot, org.freedesktop.login1.reboot−multiple−sessions, org.freedesktop.login1.reboot−ignore−inhibit, org.freedesktop.login1.halt, org.freedesktop.login1.halt−multiple−sessions, org.freedesktop.login1.halt−ignore−inhibit,

org.freedesktop.login1.suspend, org.freedesktop.login1.suspend−multiple−sessions, org.freedesktop.login1.suspend−ignore−inhibit, org.freedesktop.login1.hibernate, org.freedesktop.login1.hibernate−multiple−sessions, org.freedesktop.login1.hibernate−ignore−inhibit, respectively depending on whether there are other sessions around or active inhibits are present. **HybridSleep()** and **SuspendThenHibernate()** use the same privileges as **Hibernate()**. **SetRebootParameter()** requires org.freedesktop.login1.set−reboot−parameter.

SetRebootToFirmwareSetup requires org.freedesktop.login1.set−reboot−to−firmware−setup. **SetRebootToBootLoaderMenu** requires org.freedesktop.login1.set−reboot−to−boot−loader−menu. **SetRebootToBootLoaderEntry** requires org.freedesktop.login1.set−reboot−to−boot−loader−entry.

ScheduleShutdown and **CancelScheduledShutdown** require the same privileges (listed above) as the immediate poweroff/reboot/halt operations.

Inhibit() is protected via one of org.freedesktop.login1.inhibit−block−shutdown, org.freedesktop.login1.inhibit−delay−shutdown, org.freedesktop.login1.inhibit−block−sleep, org.freedesktop.login1.inhibit−delay−sleep, org.freedesktop.login1.inhibit−block−idle, org.freedesktop.login1.inhibit−handle−power−key, org.freedesktop.login1.inhibit−handle−suspend−key, org.freedesktop.login1.inhibit−handle−hibernate−key, org.freedesktop.login1.inhibit−handle−lid−switch depending on the lock type and mode taken.

The *interactive* boolean parameters can be used to control whether polkit should interactively ask the user for authentication credentials if required.

SEAT OBJECTS

```
node /org/freedesktop/login1/seat/seat0 {
  interface org.freedesktop.login1.Seat {
    methods:
      Terminate();
      ActivateSession(in  s session_id);
      SwitchTo(in  u vtnr);
      SwitchToNext();
      SwitchToPrevious();
    properties:
      @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
      readonly s Id = '...';
      readonly (so) ActiveSession = ...;
      @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
      readonly b CanTTY = ...;
      readonly b CanGraphical = ...;
      @org.freedesktop.DBus.Property.EmitsChangedSignal("false")
      readonly a(so) Sessions = [...];
      readonly b IdleHint = ...;
      readonly t IdleSinceHint = ...;
      readonly t IdleSinceHintMonotonic = ...;
  };
  interface org.freedesktop.DBus.Peer { ... };
  interface org.freedesktop.DBus.Introspectable { ... };
  interface org.freedesktop.DBus.Properties { ... };
};
```

Methods

Terminate() and **ActivateSession()** work similar to TerminateSeat(), ActivationSessionOnSeat() on the Manager object.

SwitchTo() switches to the session on the virtual terminal *vtnr*. **SwitchToNext()** and **SwitchToPrevious()** switch to, respectively, the next and previous sessions on the seat in the order of virtual terminals. If there is no active session, they switch to, respectively, the first and last session on the seat.

Signals

Whenever **ActiveSession**, **Sessions**, **CanGraphical**, **CanTTY**, or the idle state changes, **PropertyChanged** signals are sent out to which clients can subscribe.

Properties

The *Id* property encodes the ID of the seat.

ActiveSession encodes the currently active session if there is one. It is a structure consisting of the session id and the object path.

CanTTY encodes whether the session is suitable for text logins, and *CanGraphical* whether it is suitable for graphical sessions.

The *Sessions* property is an array of all current sessions of this seat, each encoded in a structure consisting of the ID and the object path.

The *IdleHint*, *IdleSinceHint*, and *IdleSinceHintMonotonic* properties encode the idle state, similar to the ones exposed on the Manager object, but specific for this seat.

USER OBJECTS

```
node /org/freedesktop/login1/user/_1000 {
  interface org.freedesktop.login1.User {
    methods:
      Terminate();
      Kill(in  i signal_number);
    properties:
      @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
      readonly u UID = ...;
      @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
      readonly u GID = ...;
      @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
      readonly s Name = '...';
      @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 RuntimePath = '...';
      @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
      readonly s Service = '...';
      @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
      readonly s Slice = '...';
      readonly (so) Display = ...;
      @org.freedesktop.DBus.Property.EmitsChangedSignal("false")
      readonly s State = '...';
      @org.freedesktop.DBus.Property.EmitsChangedSignal("false")
      readonly a(so) Sessions = [...];
      readonly b IdleHint = ...;
      readonly t IdleSinceHint = ...;
      readonly t IdleSinceHintMonotonic = ...;
      @org.freedesktop.DBus.Property.EmitsChangedSignal("false")
```

```
        readonly b Linger = ...;
    };
    interface org.freedesktop.DBus.Peer { ... };
    interface org.freedesktop.DBus.Introspectable { ... };
    interface org.freedesktop.DBus.Properties { ... };
};
```

Methods

Terminate() and **Kill()** work similar to the **TerminateUser()** and **KillUser()** methods on the manager object.

Signals

Whenever *Sessions* or the idle state changes, **PropertyChanged** signals are sent out to which clients can subscribe.

Properties

The *UID* and *GID* properties encode the Unix UID and primary GID of the user.

The *Name* property encodes the user name.

Timestamp and *TimestampMonotonic* encode the login time of the user in microseconds since the epoch, in the **CLOCK_REALTIME** and **CLOCK_MONOTONIC** clocks, respectively.

RuntimePath encodes the runtime path of the user, i.e. *$XDG_RUNTIME_DIR*. For details see the **XDG Basedir Specification**[3].

Service contains the unit name of the user systemd service of this user. Each logged in user is assigned a user service that runs a user systemd instance. This is usually an instance of user@.service.

Slice contains the unit name of the user systemd slice of this user. Each logged in user gets a private slice.

Display encodes which graphical session should be used as the primary UI display for the user. It is a structure encoding the session ID and the object path of the session to use.

State encodes the user state and is one of "offline", "lingering", "online", "active", or "closing". See **sd_uid_get_state**(3) for more information about the states.

Sessions is an array of structures encoding all current sessions of the user. Each structure consists of the ID and object path.

The *IdleHint*, *IdleSinceHint*, and *IdleSinceHintMonotonic* properties encode the idle hint state of the user, similar to the Manager's properties, but specific for this user.

The *Linger* property shows whether lingering is enabled for this user.

SESSION OBJECTS

```
node /org/freedesktop/login1/session/1 {
 interface org.freedesktop.login1.Session {
  methods:
   Terminate();
   Activate();
   Lock();
   Unlock();
   SetIdleHint(in  b idle);
   SetLockedHint(in  b locked);
   Kill(in  s who,
        in  i signal_number);
   TakeControl(in  b force);
   ReleaseControl();
   SetType(in  s type);
   TakeDevice(in  u major,
              in  u minor,
              out h fd,
              out b inactive);
   ReleaseDevice(in  u major,
                 in  u minor);
   PauseDeviceComplete(in  u major,
                       in  u minor);
   SetBrightness(in  s subsystem,
                 in  s name,
                 in  u brightness);
  signals:
   PauseDevice(u major,
               u minor,
               s type);
   ResumeDevice(u major,
                u minor,
                h fd);
   Lock();
   Unlock();
  properties:
   @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
   readonly s Id = '...';
   @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
   readonly (uo) User = ...;
   @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
   readonly s Name = '...';
   @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 u VTNr = ...;
   @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
   readonly (so) Seat = ...;
   @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
   readonly s TTY = '...';
   @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
```

```
        readonly s Display = '...';
        @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
        readonly b Remote = ...;
        @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
        readonly s RemoteHost = '...';
        @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
        readonly s RemoteUser = '...';
        @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
        readonly s Service = '...';
        @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
        readonly s Desktop = '...';
        @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
        readonly s Scope = '...';
        @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
        readonly u Leader = ...;
        @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
        readonly u Audit = ...;
        readonly s Type = '...';
        @org.freedesktop.DBus.Property.EmitsChangedSignal("const")
        readonly s Class = '...';
        readonly b Active = ...;
        readonly s State = '...';
        readonly b IdleHint = ...;
        readonly t IdleSinceHint = ...;
        readonly t IdleSinceHintMonotonic = ...;
        readonly b LockedHint = ...;
    };
    interface org.freedesktop.DBus.Peer { ... };
    interface org.freedesktop.DBus.Introspectable { ... };
    interface org.freedesktop.DBus.Properties { ... };
};
```

Methods

Terminate(), **Activate()**, **Lock()**, **Unlock()**, and **Kill()** work similarly to the respective calls on the Manager object.

SetIdleHint() is called by the session object to update the idle state of the session whenever it changes.

TakeControl() allows a process to take exclusive managed device access−control for that session. Only one D−Bus connection can be a controller for a given session at any time. If the *force* argument is set (root only), an existing controller is kicked out and replaced. Otherwise, this method fails if there is already a controller. Note that this method is limited to D−Bus users with the effective UID set to the user of the session or root.

ReleaseControl() drops control of a given session. Closing the D−Bus connection implicitly releases control as well. See **TakeControl()** for more information. This method also releases all devices for which the controller requested ownership via **TakeDevice()**.

SetType() allows the type of the session to be changed dynamically. It can only be called by session's current controller. If **TakeControl()** has not been called, this method will fail. In addition, the session type will be reset to its original value once control is released, either by calling **ReleaseControl()** or closing the D−Bus connection. This should help prevent a session from entering an inconsistent state, for example if the controller crashes. The only argument *type* is the new session type.

TakeDevice() allows a session controller to get a file descriptor for a specific device. Pass in the major and minor numbers of the character device and systemd−logind will return a file descriptor for the device. Only a limited set of device−types is currently supported (but may be extended). systemd−logind automatically mutes the file descriptor if the session is inactive and resumes it once the session is activated again. This guarantees that a session can only access session devices if the session is active. Note that this revoke/resume mechanism is asynchronous and may happen at any given time. This only works on devices that are attached to the seat of the given session. A process is not required to have direct access to the device node. systemd−logind only requires you to be the active session controller (see **TakeControl()**). Also note that any device can only be requested once. As long as you don't release it, further **TakeDevice()** calls will fail.

ReleaseDevice() releases a device again (see **TakeDevice()**). This is also implicitly done by **ReleaseControl()** or when closing the D−Bus connection.

PauseDeviceComplete() allows a session controller to synchronously pause a device after receiving a **PauseDevice("pause")** signal. Forced signals (or after an internal timeout) are automatically completed by systemd−logind asynchronously.

SetLockedHint() may be used to set the "locked hint" to *locked*, i.e. information whether the session is locked. This is intended to be used by the desktop environment to tell **systemd−logind** when the session is locked and unlocked.

SetBrightness() may be used to set the display brightness. This is intended to be used by the desktop environment and allows unprivileged programs to access hardware settings in a controlled way. The *subsystem* parameter specifies a kernel subsystem, either "backlight" or "leds". The *name* parameter specifies a device name under the specified subsystem. The *brightness* parameter specifies the brightness. The range is defined by individual drivers, see /sys/class/*subsystem*/*name*/max_brightness.

Signals

The active session controller exclusively gets **PauseDevice** and **ResumeDevice** events for any device it requested via **TakeDevice()**. They notify the controller whenever a device is paused or resumed. A device is never resumed if its session is inactive. Also note that **PauseDevice** signals are sent before the **PropertyChanged** signal for the **Active** state. The inverse is true for **ResumeDevice**. A device may remain paused for unknown reasons even though the Session is active.

A **PauseDevice** signal carries the major and minor numbers and a string describing the type as arguments. **force** means the device was already paused by systemd−logind and the signal is only an asynchronous notification. **pause** means systemd−logind grants you a limited amount of time to pause the device. You must respond to this via **PauseDeviceComplete()**. This synchronous pausing mechanism is used for

backwards−compatibility to VTs and systemd−logind is free to not make use of it. It is also free to send a forced **PauseDevice** if you don't respond in a timely manner (or for any other reason). **gone** means the device was unplugged from the system and you will no longer get any notifications about it. There is no need to call **ReleaseDevice()**. You may call **TakeDevice()** again if a new device is assigned the major+minor combination.

ResumeDevice is sent whenever a session is active and a device is resumed. It carries the major/minor numbers as arguments and provides a new open file descriptor. You should switch to the new descriptor and close the old one. They are not guaranteed to have the same underlying open file descriptor in the kernel (except for a limited set of device types).

Whenever **Active** or the idle state changes, **PropertyChanged** signals are sent out to which clients can subscribe.

Lock/**Unlock** is sent when the session is asked to be screen−locked/unlocked. A session manager of the session should listen to this signal and act accordingly. This signal is sent out as a result of the **Lock()** and **Unlock()** methods, respectively.

Properties

Id encodes the session ID.

User encodes the user ID of the user this session belongs to. This is a structure consisting of the Unix UID and the object path.

Name encodes the user name.

Timestamp and *TimestampMonotonic* encode the microseconds since the epoch when the session was created, in **CLOCK_REALTIME** or **CLOCK_MONOTONIC**, respectively.

VTNr encodes the virtual terminal number of the session if there is any, 0 otherwise.

Seat encodes the seat this session belongs to if there is any. This is a structure consisting of the ID and the seat object path.

TTY encodes the kernel TTY path of the session if this is a text login. If not this is an empty string.

Display encodes the X11 display name if this is a graphical login. If not, this is an empty string.

Remote encodes whether the session is local or remote.

RemoteHost and *RemoteUser* encode the remote host and user if this is a remote session, or an empty string otherwise.

Service encodes the PAM service name that registered the session.

Desktop describes the desktop environment running in the session (if known).

Scope contains the systemd scope unit name of this session.

Leader encodes the PID of the process that registered the session.

Audit encodes the Kernel Audit session ID of the session if auditing is available.

Type encodes the session type. It's one of "unspecified" (for cron PAM sessions and suchlike), "tty" (for text logins) or "x11"/"mir"/"wayland" (for graphical logins).

Class encodes the session class. It's one of "user" (for normal user sessions), "greeter" (for display manager pseudo−sessions), or "lock−screen" (for display lock screens).

Active is a boolean that is true if the session is active, i.e. currently in the foreground. This field is semi−redundant due to *State*.

State encodes the session state and one of "online", "active", or "closing". See **sd_session_get_state**(3) for more information about the states.

IdleHint, *IdleSinceHint*, and *IdleSinceHintMonotonic* encapsulate the idle hint state of this session, similarly to how the respective properties on the manager object do it for the whole system.

LockedHint shows the locked hint state of this session, as set by the **SetLockedHint()** method described

above.

EXAMPLES

Example 1. Introspect org.freedesktop.login1.Manager on the bus

```
$ gdbus introspect --system --dest org.freedesktop.login1 \
  --object-path /org/freedesktop/login1
```

Example 2. Introspect org.freedesktop.login1.Seat on the bus

```
$ gdbus introspect --system --dest org.freedesktop.login1 \
 --object-path /org/freedesktop/login1/seat/seat0
```

Example 3. Introspect org.freedesktop.login1.User on the bus

```
$ gdbus introspect --system --dest org.freedesktop.login1 \
  --object-path /org/freedesktop/login1/user/_1000
```

Example 4. Introspect org.freedesktop.login1.Session on the bus

```
$ gdbus introspect --system --dest org.freedesktop.login1 \
  --object-path /org/freedesktop/login1/session/45
```

VERSIONING

These D-Bus interfaces follow **the usual interface versioning guidelines**[4].

NOTES

1. polkit
   https://www.freedesktop.org/software/polkit/docs/latest/
2. Inhibitor Locks
   https://www.freedesktop.org/wiki/Software/systemd/inhibit
3. XDG Basedir Specification
   https://specifications.freedesktop.org/basedir-spec/basedir-spec-latest.html
4. the usual interface versioning guidelines
   http://0pointer.de/blog/projects/versioning-dbus.html