logiciel, BROUILLON

Systemd

Pré-requis

Installation

Configuration

Options du fichier *.service

section [Unit]

Description=
: Chaîne libre décrivant le service. Exemple : “serveur web Apache2”. !!

Documentation=
: liste d'URIs (“http:*”, “https:*”, “file:”, “info:”, “man:”) séparées par des virgules, par ordre de pertinence, référençant une documentation pour ce service ou sa configuration. !!

Requires=
: dépendances d'autres services. Si ce service est activé, les autres le seront aussi. !!

RequiresOverridable=
: Similar to Requires=. Dependencies listed in RequiresOverridable= which cannot be fulfilled or fail to start are ignored if the startup was explicitly requested by the user. If the start-up was pulled in indirectly by some dependency or automatic start-up of units that is not requested by the user, this dependency must be fulfilled and otherwise the transaction fails. Hence, this option may be used to configure dependencies that are normally honored unless the user explicitly starts up the unit, in which case whether they failed or not is irrelevant. !!

Requisite=, RequisiteOverridable=
: Similar to Requires= and RequiresOverridable=, respectively. However, if the units listed here are not started already, they will not be started and the transaction will fail immediately. !!

Wants=
: A weaker version of Requires=. Units listed in this option will be started if the configuring unit is. However, if the listed units fail to start or cannot be added to the transaction, this has no impact on the validity of the transaction as a whole. This is the recommended way to hook start-up of one unit to the start-up of another unit.
Note that dependencies of this type may also be configured outside of the unit configuration file by adding symlinks to a .wants/ directory accompanying the unit file. For details, see above. !!

BindsTo=
: Configures requirement dependencies, very similar in style to Requires=, however in addition to this behavior, it also declares that this unit is stopped when any of the units

listed suddenly disappears. Units can suddenly, unexpectedly disappear if a service terminates on its own choice, a device is unplugged or a mount point unmounted without involvement of systemd. !!

PartOf=

Configures dependencies similar to Requires=, but limited to stopping and restarting of units. When systemd stops or restarts the units listed here, the action is propagated to this unit. Note that this is a one-way dependency — changes to this unit do not affect the listed units. !!

Conflicts=

A space-separated list of unit names. Configures negative requirement dependencies. If a unit has a Conflicts= setting on another unit, starting the former will stop the latter and vice versa. Note that this setting is independent of and orthogonal to the After= and Before= ordering dependencies.
If a unit A that conflicts with a unit B is scheduled to be started at the same time as B, the transaction will either fail (in case both are required part of the transaction) or be modified to be fixed (in case one or both jobs are not a required part of the transaction). In the latter case, the job that is not the required will be removed, or in case both are not required, the unit that conflicts will be started and the unit that is conflicted is stopped. !!

Before=, After=

A space-separated list of unit names. Configures ordering dependencies between units. If a unit foo.service contains a setting Before=bar.service and both units are being started, bar.service's start-up is delayed until foo.service is started up. Note that this setting is independent of and orthogonal to the requirement dependencies as configured by Requires=. It is a common pattern to include a unit name in both the After= and Requires= option, in which case the unit listed will be started before the unit that is configured with these options. This option may be specified more than once, in which case ordering dependencies for all listed names are created. After= is the inverse of Before=, i.e. while After= ensures that the configured unit is started after the listed unit finished starting up, Before= ensures the opposite, i.e. that the configured unit is fully started up before the listed unit is started. Note that when two units with an ordering dependency between them are shut down, the inverse of the start-up order is applied. i.e. if a unit is configured with After= on another unit, the former is stopped before the latter if both are shut down. If one unit with an ordering dependency on another unit is shut down while the latter is started up, the shut down is ordered before the start-up regardless of whether the ordering dependency is actually of type After= or Before=. If two units have no ordering dependencies between them, they are shut down or started up simultaneously, and no ordering takes place. !!

OnFailure=

A space-separated list of one or more units that are activated when this unit enters the “failed” state. !!

PropagatesReloadTo=, ReloadPropagatedFrom=

A space-separated list of one or more units where reload requests on this unit will be propagated to, or reload requests on the other unit will be propagated to this unit, respectively. Issuing a reload request on a unit will automatically also enqueue a reload request on all units that the reload request shall be propagated to via these two settings. !!

JoinsNamespaceOf=

For units that start processes (such as service units), lists one or more other units whose network and/or temporary file namespace to join. This only applies to unit types which support the PrivateNetwork= and PrivateTmp= directives (see systemd.exec(5) for details). If a unit that has this setting set is started, its processes will see the same /tmp,

/tmp/var and network namespace as one listed unit that is started. If multiple listed units are already started, it is not defined which namespace is joined. Note that this setting only has an effect if PrivateNetwork= and/or PrivateTmp= is enabled for both the unit that joins the namespace and the unit whose namespace is joined. !!

RequiresMountsFor=

Takes a space-separated list of absolute paths. Automatically adds dependencies of type Requires= and After= for all mount units required to access the specified path.
Mount points marked with noauto are not mounted automatically and will be ignored for the purposes of this option. If such a mount should be a requirement for this unit, direct dependencies on the mount units may be added (Requires= and After= or some other combination). !!

OnFailureJobMode=

Takes a value of “fail”, “replace”, “replace-irreversibly”, “isolate”, “flush”, “ignore-dependencies” or “ignore-requirements”. Defaults to “replace”. Specifies how the units listed in OnFailure= will be enqueued. See systemctl(1)'s –job-mode= option for details on the possible values. If this is set to “isolate”, only a single unit may be listed in OnFailure=.. !!

IgnoreOnIsolate=

Takes a boolean argument. If true, this unit will not be stopped when isolating another unit. Defaults to false. !!

IgnoreOnSnapshot=

Takes a boolean argument. If true, this unit will not be included in snapshots. Defaults to true for device and snapshot units, false for the others. !!

StopWhenUnneeded=

Takes a boolean argument. If true, this unit will be stopped when it is no longer used. Note that in order to minimize the work to be executed, systemd will not stop units by default unless they are conflicting with other units, or the user explicitly requested their shut down. If this option is set, a unit will be automatically cleaned up if no other active unit requires it. !!

Defaults to false.
!!

RefuseManualStart=, RefuseManualStop=

Takes a boolean argument. If true, this unit can only be activated or deactivated indirectly. In this case, explicit start-up or termination requested by the user is denied, however if it is started or stopped as a dependency of another unit, start-up or termination will succeed. This is mostly a safety feature to ensure that the user does not accidentally activate units that are not intended to be activated explicitly, and not accidentally deactivate units that are not intended to be deactivated. These options default to false. !!

AllowIsolate=

Takes a boolean argument. If true, this unit may be used with the systemctl isolate command. Otherwise, this will be refused. It probably is a good idea to leave this disabled except for target units that shall be used similar to runlevels in SysV init systems, just as a precaution to avoid unusable system states. This option defaults to false. !!

DefaultDependencies=

Takes a boolean argument. If true, (the default), a few default dependencies will implicitly be created for the unit. The actual dependencies created depend on the unit type. For example, for service units, these dependencies ensure that the service is started only after basic system initialization is completed and is properly terminated on

system shutdown. See the respective man pages for details. Generally, only services involved with early boot or late shutdown should set this option to false. It is highly recommended to leave this option enabled for the majority of common units. If set to false, this option does not disable all implicit dependencies, just non-essential ones. !!

JobTimeoutSec=, JobTimeoutAction=, JobTimeoutRebootArgument=

When a job for this unit is queued a time-out may be configured. If this time limit is reached, the job will be cancelled, the unit however will not change state or even enter the “failed” mode. This value defaults to 0 (job timeouts disabled), except for device units. NB: this timeout is independent from any unit-specific timeout (for example, the timeout set with TimeoutStartSec= in service units) as the job timeout has no effect on the unit itself, only on the job that might be pending for it. Or in other words: unit-specific timeouts are useful to abort unit state changes, and revert them. The job timeout set with this option however is useful to abort only the job waiting for the unit state to change. !!

JobTimeoutAction=

optionally configures an additional action to take when the time-out is hit. It takes the same values as the per-service StartLimitAction= setting, see systemd.service(5) for details. Defaults to none. !!

JobTimeoutRebootArgument=

configures an optional reboot string to pass to the reboot(2) system call. !!

ConditionArchitecture=, ConditionVirtualization=, ConditionHost=, ConditionKernelCommandLine=, ConditionSecurity=, ConditionCapability=, ConditionACPower=, ConditionNeedsUpdate=, ConditionFirstBoot=, ConditionPathExists=, ConditionPathExistsGlob=, ConditionPathIsDirectory=, ConditionPathIsSymbolicLink=, ConditionPathIsMountPoint=, ConditionPathIsReadWrite=, ConditionDirectoryNotEmpty=, ConditionFileNotEmpty=, ConditionFileIsExecutable=

Before starting a unit verify that the specified condition is true. If it is not true, the starting of the unit will be skipped, however all ordering dependencies of it are still respected. A failing condition will not result in the unit being moved into a failure state. The condition is checked at the time the queued start job is to be executed. !!

ConditionArchitecture=

may be used to check whether the system is running on a specific architecture. Takes one of x86, x86-64, ppc, ppc-le, ppc64, ppc64-le, ia64, parisc, parisc64, s390, s390x, sparc, sparc64, mips, mips-le, mips64, mips64-le, alpha, arm, arm-be, arm64, arm64-be, sh, sh64, m86k, tilegx, cris to test against a specific architecture. The architecture is determined from the information returned by uname(2) and is thus subject to personality(2). Note that a Personality= setting in the same unit file has no effect on this condition. A special architecture name native is mapped to the architecture the system manager itself is compiled for. The test may be negated by prepending an exclamation mark. !!

ConditionVirtualization=

may be used to check whether the system is executed in a virtualized environment and optionally test whether it is a specific implementation. Takes either boolean value to check if being executed in any virtualized environment, or one of vm and container to test against a generic type of virtualization solution, or one of qemu, kvm, zvm, vmware, microsoft, oracle, xen, bochs, uml, openvz, lxc, lxc-libvirt, systemd-nspawn, docker to test against a specific implementation. See systemd-detect-virt(1) for a full list of

known virtualization technologies and their identifiers. If multiple virtualization technologies are nested, only the innermost is considered. The test may be negated by prepending an exclamation mark. !!

ConditionHost=

may be used to match against the hostname or machine ID of the host. This either takes a hostname string (optionally with shell style globs) which is tested against the locally set hostname as returned by gethostname(2), or a machine ID formatted as string (see machine-id(5)). The test may be negated by prepending an exclamation mark.

ConditionKernelCommandLine=

may be used to check whether a specific kernel command line option is set (or if prefixed with the exclamation mark unset). The argument must either be a single word, or an assignment (i.e. two words, separated “=”). In the former case the kernel command line is searched for the word appearing as is, or as left hand side of an assignment. In the latter case, the exact assignment is looked for with right and left hand side matching.

ConditionSecurity=

may be used to check whether the given security module is enabled on the system. Currently the recognized values values are selinux, apparmor, ima, smack and audit. The test may be negated by prepending an exclamation mark. !!

ConditionCapability=

may be used to check whether the given capability exists in the capability bounding set of the service manager (i.e. this does not check whether capability is actually available in the permitted or effective sets, see capabilities(7) for details). Pass a capability name such as “CAP_MKNOD”, possibly prefixed with an exclamation mark to negate the check. !!

ConditionACPower=

may be used to check whether the system has AC power, or is exclusively battery powered at the time of activation of the unit. This takes a boolean argument. If set to true, the condition will hold only if at least one AC connector of the system is connected to a power source, or if no AC connectors are known. Conversely, if set to false, the condition will hold only if there is at least one AC connector known and all AC connectors are disconnected from a power source. !!

ConditionNeedsUpdate=

takes one of /var or /etc as argument, possibly prefixed with a “!” (for inverting the condition). This condition may be used to conditionalize units on whether the specified directory requires an update because /usr's modification time is newer than the stamp file .updated in the specified directory. This is useful to implement offline updates of the vendor operating system resources in /usr that require updating of /etc or /var on the next following boot. Units making use of this condition should order themselves before systemd-update-done.service(8), to make sure they run before the stamp files's modification time gets reset indicating a completed update. !!

ConditionFirstBoot=

takes a boolean argument. This condition may be used to conditionalize units on whether the system is booting up with an unpopulated /etc directory. This may be used to populate /etc on the first boot after factory reset, or when a new system instances boots up for the first time. !!

With

ConditionPathExists= a file existence condition is checked before a unit is started. If the specified absolute path name does not exist, the condition will fail. If the absolute path name passed to ConditionPathExists= is prefixed with an exclamation mark (“!”), the test is negated, and the unit is only started if the path does not exist. !!

ConditionPathExistsGlob=
: is similar to ConditionPathExists=, but checks for the existence of at least one file or directory matching the specified globbing pattern. !!

ConditionPathIsDirectory=
: is similar to ConditionPathExists= but verifies whether a certain path exists and is a directory. !!

ConditionPathIsSymbolicLink=
: is similar to ConditionPathExists= but verifies whether a certain path exists and is a symbolic link. !!

ConditionPathIsMountPoint=
: is similar to ConditionPathExists= but verifies whether a certain path exists and is a mount point. !!

ConditionPathIsReadWrite=
: is similar to ConditionPathExists= but verifies whether the underlying file system is readable and writable (i.e. not mounted read-only). !!

ConditionDirectoryNotEmpty=
: is similar to ConditionPathExists= but verifies whether a certain path exists and is a non-empty directory. !!

ConditionFileNotEmpty=
: is similar to ConditionPathExists= but verifies whether a certain path exists and refers to a regular file with a non-zero size. !!

ConditionFileIsExecutable=
: is similar to ConditionPathExists= but verifies whether a certain path exists, is a regular file and marked executable. !!

If multiple conditions are specified, the unit will be executed if all of them apply (i.e. a logical AND is applied). Condition checks can be prefixed with a pipe symbol (|) in which case a condition becomes a triggering condition. If at least one triggering condition is defined for a unit, then the unit will be executed if at least one of the triggering conditions apply and all of the non-triggering conditions. If you prefix an argument with the pipe symbol and an exclamation mark, the pipe symbol must be passed first, the exclamation second. Except for ConditionPathIsSymbolicLink=, all path checks follow symlinks. If any of these options is assigned the empty string, the list of conditions is reset completely, all previous condition settings (of any kind) will have no effect. !!

AssertArchitecture=, AssertVirtualization=, AssertHost=, AssertKernelCommandLine=, AssertSecurity=, AssertCapability=, AssertACPower=, AssertNeedsUpdate=, AssertFirstBoot=, AssertPathExists=, AssertPathExistsGlob=, AssertPathIsDirectory=, AssertPathIsSymbolicLink=, AssertPathIsMountPoint=, AssertPathIsReadWrite=, AssertDirectoryNotEmpty=, AssertFileNotEmpty=, AssertFileIsExecutable=
: Similar to the ConditionArchitecture=, ConditionVirtualization=, … condition settings described above these settings add assertion checks to the start-up of the unit. However, unlike the conditions settings any assertion setting that is not met results in failure of the start job it was triggered by. !!

SourcePath=
: A path to a configuration file this unit has been generated from. This is primarily useful for implementation of generator tools that convert configuration from an external configuration file format into native unit files. This functionality should not be used in

normal units. !!

NetClass=

Configures a network class number to assign to the unit. This value will be set to the “net_cls.class_id” property of the “net_cls” cgroup of the unit. The directive accepts a numerical value (for fixed number assignment) and the keyword “auto” (for dynamic allocation). Network traffic of all processes inside the unit will have the network class ID assigned by the kernel. Also see the kernel docs for net_cls controller and systemd.resource-control(5). !!

Section [Service]

```
Type=

    Configures the process start-up type for this service unit. One of
simple, forking, oneshot, dbus, notify or idle.

    If set to simple (the default if neither Type= nor BusName=, but
ExecStart= are specified), it is expected that the process configured with
ExecStart= is the main process of the service. In this mode, if the process
offers functionality to other processes on the system, its communication
channels should be installed before the daemon is started up (e.g. sockets
set up by systemd, via socket activation), as systemd will immediately
proceed starting follow-up units.

    If set to forking, it is expected that the process configured with
ExecStart= will call fork() as part of its start-up. The parent process is
expected to exit when start-up is complete and all communication channels
are set up. The child continues to run as the main daemon process. This is
the behavior of traditional UNIX daemons. If this setting is used, it is
recommended to also use the PIDFile= option, so that systemd can identify
the main process of the daemon. systemd will proceed with starting follow-up
units as soon as the parent process exits.

    Behavior of oneshot is similar to simple; however, it is expected that
the process has to exit before systemd starts follow-up units.
RemainAfterExit= is particularly useful for this type of service. This is
the implied default if neither Type= or ExecStart= are specified.

    Behavior of dbus is similar to simple; however, it is expected that the
daemon acquires a name on the D-Bus bus, as configured by BusName=. systemd
will proceed with starting follow-up units after the D-Bus bus name has been
acquired. Service units with this option configured implicitly gain
dependencies on the dbus.socket unit. This type is the default if BusName=
is specified.

    Behavior of notify is similar to simple; however, it is expected that
the daemon sends a notification message via sd_notify(3) or an equivalent
call when it has finished starting up. systemd will proceed with starting
follow-up units after this notification message has been sent. If this
option is used, NotifyAccess= (see below) should be set to open access to
```

the notification socket provided by systemd. If NotifyAccess= is not set, it will be implicitly set to main. Note that currently Type=notify will not work if used in combination with PrivateNetwork=yes.

Behavior of idle is very similar to simple; however, actual execution of the service binary is delayed until all jobs are dispatched. This may be used to avoid interleaving of output of shell services with the status output on the console.
RemainAfterExit=

Takes a boolean value that specifies whether the service shall be considered active even when all its processes exited. Defaults to no.
GuessMainPID=

Takes a boolean value that specifies whether systemd should try to guess the main PID of a service if it cannot be determined reliably. This option is ignored unless Type=forking is set and PIDFile= is unset because for the other types or with an explicitly configured PID file, the main PID is always known. The guessing algorithm might come to incorrect conclusions if a daemon consists of more than one process. If the main PID cannot be determined, failure detection and automatic restarting of a service will not work reliably. Defaults to yes.
PIDFile=

Takes an absolute file name pointing to the PID file of this daemon. Use of this option is recommended for services where Type= is set to forking. systemd will read the PID of the main process of the daemon after start-up of the service. systemd will not write to the file configured here, although it will remove the file after the service has shut down if it still exists.
BusName=

Takes a D-Bus bus name that this service is reachable as. This option is mandatory for services where Type= is set to dbus.
BusPolicy=

If specified, a custom kdbus endpoint will be created and installed as the default bus node for the service. Such a custom endpoint can hold an own set of policy rules that are enforced on top of the bus-wide ones. The custom endpoint is named after the service it was created for, and its node will be bind-mounted over the default bus node location, so the service can only access the bus through its own endpoint. Note that custom bus endpoints default to a 'deny all' policy. Hence, if at least one BusPolicy= directive is given, you have to make sure to add explicit rules for everything the service should be able to do.

The value of this directive is comprised of two parts; the bus name, and a verb to specify to granted access, which is one of see, talk, or own. talk implies see, and own implies both talk and see. If multiple access levels are specified for the same bus name, the most powerful one takes effect.

Examples:

```
    BusPolicy=org.freedesktop.systemd1 talk

    BusPolicy=org.foo.bar see

    This option is only available on kdbus enabled systems.
ExecStart=

    Commands with their arguments that are executed when this service is
started. The value is split into zero or more command lines is according to
the rules described below (see section "Command Lines" below).

    When Type= is not oneshot, only one command may and must be given. When
Type=oneshot is used, zero or more commands may be specified. This can be
specified by providing multiple command lines in the same directive, or
alternatively, this directive may be specified more than once with the same
effect. If the empty string is assigned to this option, the list of commands
to start is reset, prior assignments of this option will have no effect. If
no ExecStart= is specified, then the service must have RemainAfterExit=yes
set.

    For each of the specified commands, the first argument must be an
absolute path to an executable. Optionally, if this file name is prefixed
with "@", the second token will be passed as "argv[0]" to the executed
process, followed by the further arguments specified. If the absolute
filename is prefixed with "-", an exit code of the command normally
considered a failure (i.e. non-zero exit status or abnormal exit due to
signal) is ignored and considered success. If both "-" and "@" are used,
they can appear in either order.

    If more than one command is specified, the commands are invoked
sequentially in the order they appear in the unit file. If one of the
commands fails (and is not prefixed with "-"), other lines are not executed,
and the unit is considered failed.

    Unless Type=forking is set, the process started via this command line
will be considered the main process of the daemon.
ExecStartPre=, ExecStartPost=

    Additional commands that are executed before or after the command in
ExecStart=, respectively. Syntax is the same as for ExecStart=, except that
multiple command lines are allowed and the commands are executed one after
the other, serially.

    If any of those commands (not prefixed with "-") fail, the rest are not
executed and the unit is considered failed.

    ExecStart= commands are only run after all ExecStartPre= commands that
were not prefixed with a "-" exit successfully.

    ExecStartPost= commands are only run after the service has started, as
determined by Type= (i.e. The process has been started for Type=simple or
```

```
Type=idle, the process exits successfully for Type=oneshot, the initial
process exits successfully for Type=forking, "READY=1" is sent for
Type=notify, or the BusName= has been taken for Type=dbus).

    Note that ExecStartPre= may not be used to start long-running processes.
All processes forked off by processes invoked via ExecStartPre= will be
killed before the next service process is run.
ExecReload=

    Commands to execute to trigger a configuration reload in the service.
This argument takes multiple command lines, following the same scheme as
described for ExecStart= above. Use of this setting is optional. Specifier
and environment variable substitution is supported here following the same
scheme as for ExecStart=.

    One additional, special environment variable is set: if known, $MAINPID
is set to the main process of the daemon, and may be used for command lines
like the following:

    /bin/kill -HUP $MAINPID

    Note however that reloading a daemon by sending a signal (as with the
example line above) is usually not a good choice, because this is an
asynchronous operation and hence not suitable to order reloads of multiple
services against each other. It is strongly recommended to set ExecReload=
to a command that not only triggers a configuration reload of the daemon,
but also synchronously waits for it to complete.
ExecStop=

    Commands to execute to stop the service started via ExecStart=. This
argument takes multiple command lines, following the same scheme as
described for ExecStart= above. Use of this setting is optional. After the
commands configured in this option are run, all processes remaining for a
service are terminated according to the KillMode= setting (see
systemd.kill(5)). If this option is not specified, the process is terminated
by sending the signal specified in KillSignal= when service stop is
requested. Specifier and environment variable substitution is supported
(including $MAINPID, see above).

    Note that it is usually not sufficient to specify a command for this
setting that only asks the service to terminate (for example by queuing some
form of termination signal for it), but does not wait for it to do so. Since
the remaining processes of the services are killed using SIGKILL immediately
after the command exited this would not result in a clean stop. The
specified command should hence be a synchronous operation, not an
asynchronous one.
ExecStopPost=

    Additional commands that are executed after the service was stopped.
This includes cases where the commands configured in ExecStop= were used,
where the service does not have any ExecStop= defined, or where the service
```

exited unexpectedly. This argument takes multiple command lines, following the same scheme as described for ExecStart=. Use of these settings is optional. Specifier and environment variable substitution is supported.
RestartSec=

Configures the time to sleep before restarting a service (as configured with Restart=). Takes a unit-less value in seconds, or a time span value such as "5min 20s". Defaults to 100ms.
TimeoutStartSec=

Configures the time to wait for start-up. If a daemon service does not signal start-up completion within the configured time, the service will be considered failed and will be shut down again. Takes a unit-less value in seconds, or a time span value such as "5min 20s". Pass "0" to disable the timeout logic. Defaults to DefaultTimeoutStartSec= from the manager configuration file, except when Type=oneshot is used, in which case the timeout is disabled by default (see systemd-system.conf(5)).
TimeoutStopSec=

Configures the time to wait for stop. If a service is asked to stop, but does not terminate in the specified time, it will be terminated forcibly via SIGTERM, and after another timeout of equal duration with SIGKILL (see KillMode= in systemd.kill(5)). Takes a unit-less value in seconds, or a time span value such as "5min 20s". Pass "0" to disable the timeout logic. Defaults to DefaultTimeoutStopSec= from the manager configuration file (see systemd-system.conf(5)).
TimeoutSec=

A shorthand for configuring both TimeoutStartSec= and TimeoutStopSec= to the specified value.
WatchdogSec=

Configures the watchdog timeout for a service. The watchdog is activated when the start-up is completed. The service must call sd_notify(3) regularly with "WATCHDOG=1" (i.e. the "keep-alive ping"). If the time between two such calls is larger than the configured time, then the service is placed in a failed state and it will be terminated with SIGABRT. By setting Restart= to on-failure or always, the service will be automatically restarted. The time configured here will be passed to the executed service process in the WATCHDOG_USEC= environment variable. This allows daemons to automatically enable the keep-alive pinging logic if watchdog support is enabled for the service. If this option is used, NotifyAccess= (see below) should be set to open access to the notification socket provided by systemd. If NotifyAccess= is not set, it will be implicitly set to main. Defaults to 0, which disables this feature.
Restart=

Configures whether the service shall be restarted when the service process exits, is killed, or a timeout is reached. The service process may be the main service process, but it may also be one of the processes specified with ExecStartPre=, ExecStartPost=, ExecStop=, ExecStopPost=, or

ExecReload=. When the death of the process is a result of systemd operation (e.g. service stop or restart), the service will not be restarted. Timeouts include missing the watchdog "keep-alive ping" deadline and a service start, reload, and stop operation timeouts.

Takes one of no, on-success, on-failure, on-abnormal, on-watchdog, on-abort, or always. If set to no (the default), the service will not be restarted. If set to on-success, it will be restarted only when the service process exits cleanly. In this context, a clean exit means an exit code of 0, or one of the signals SIGHUP, SIGINT, SIGTERM or SIGPIPE, and additionally, exit statuses and signals specified in SuccessExitStatus=. If set to on-failure, the service will be restarted when the process exits with a non-zero exit code, is terminated by a signal (including on core dump, but excluding the aforementioned four signals), when an operation (such as service reload) times out, and when the configured watchdog timeout is triggered. If set to on-abnormal, the service will be restarted when the process is terminated by a signal (including on core dump, excluding the aforementioned four signals), when an operation times out, or when the watchdog timeout is triggered. If set to on-abort, the service will be restarted only if the service process exits due to an uncaught signal not specified as a clean exit status. If set to on-watchdog, the service will be restarted only if the watchdog timeout for the service expires. If set to always, the service will be restarted regardless of whether it exited cleanly or not, got terminated abnormally by a signal, or hit a timeout.

Table 1. Exit causes and the effect of the Restart= settings on them

Restart settings/Exit causes	no	always	on-success	on-failure	on-abnormal	on-abort	on-watchdog
Clean exit code or signal		X	X				
Unclean exit code		X		X			
Unclean signal		X		X	X	X	
Timeout		X		X	X		
Watchdog		X		X	X		X

As exceptions to the setting above the service will not be restarted if the exit code or signal is specified in RestartPreventExitStatus= (see below). Also, the services will always be restarted if the exit code or signal is specified in RestartForceExitStatus= (see below).

Setting this to on-failure is the recommended choice for long-running services, in order to increase reliability by attempting automatic recovery from errors. For services that shall be able to terminate on their own choice (and avoid immediate restarting), on-abnormal is an alternative choice.

SuccessExitStatus=

Takes a list of exit status definitions that when returned by the main service process will be considered successful termination, in addition to the normal successful exit code 0 and the signals SIGHUP, SIGINT, SIGTERM, and SIGPIPE. Exit status definitions can either be numeric exit codes or termination signal names, separated by spaces. For example:

```
    SuccessExitStatus=1 2 8
            SIGKILL

    ensures that exit codes 1, 2, 8 and the termination signal SIGKILL are
considered clean service terminations.

    Note that if a process has a signal handler installed and exits by
calling _exit(2) in response to a signal, the information about the signal
is lost. Programs should instead perform cleanup and kill themselves with
the same signal instead. See Proper handling of SIGINT/SIGQUIT — How to be a
proper program.

    This option may appear more than once, in which case the list of
successful exit statuses is merged. If the empty string is assigned to this
option, the list is reset, all prior assignments of this option will have no
effect.
RestartPreventExitStatus=

    Takes a list of exit status definitions that when returned by the main
service process will prevent automatic service restarts, regardless of the
restart setting configured with Restart=. Exit status definitions can either
be numeric exit codes or termination signal names, and are separated by
spaces. Defaults to the empty list, so that, by default, no exit status is
excluded from the configured restart logic. For example:

    RestartPreventExitStatus=1 6
            SIGABRT

    ensures that exit codes 1 and 6 and the termination signal SIGABRT will
not result in automatic service restarting. This option may appear more than
once, in which case the list of restart-preventing statuses is merged. If
the empty string is assigned to this option, the list is reset and all prior
assignments of this option will have no effect.
RestartForceExitStatus=

    Takes a list of exit status definitions that when returned by the main
service process will force automatic service restarts, regardless of the
restart setting configured with Restart=. The argument format is similar to
RestartPreventExitStatus=.
PermissionsStartOnly=

    Takes a boolean argument. If true, the permission-related execution
options, as configured with User= and similar options (see systemd.exec(5)
for more information), are only applied to the process started with
ExecStart=, and not to the various other ExecStartPre=, ExecStartPost=,
ExecReload=, ExecStop=, and ExecStopPost= commands. If false, the setting is
applied to all configured commands the same way. Defaults to false.
RootDirectoryStartOnly=

    Takes a boolean argument. If true, the root directory, as configured
with the RootDirectory= option (see systemd.exec(5) for more information),
```

```
is only applied to the process started with ExecStart=, and not to the
various other ExecStartPre=, ExecStartPost=, ExecReload=, ExecStop=, and
ExecStopPost= commands. If false, the setting is applied to all configured
commands the same way. Defaults to false.
NonBlocking=

    Set the O_NONBLOCK flag for all file descriptors passed via socket-based
activation. If true, all file descriptors >= 3 (i.e. all except stdin,
stdout, and stderr) will have the O_NONBLOCK flag set and hence are in non-
blocking mode. This option is only useful in conjunction with a socket unit,
as described in systemd.socket(5). Defaults to false.
NotifyAccess=

    Controls access to the service status notification socket, as accessible
via the sd_notify(3) call. Takes one of none (the default), main or all. If
none, no daemon status updates are accepted from the service processes, all
status update messages are ignored. If main, only service updates sent from
the main process of the service are accepted. If all, all services updates
from all members of the service's control group are accepted. This option
should be set to open access to the notification socket when using
Type=notify or WatchdogSec= (see above). If those options are used but
NotifyAccess= is not configured, it will be implicitly set to main.
Sockets=

    Specifies the name of the socket units this service shall inherit socket
file descriptors from when the service is started. Normally it should not be
necessary to use this setting as all socket file descriptors whose unit
shares the same name as the service (subject to the different unit name
suffix of course) are passed to the spawned process.

    Note that the same socket file descriptors may be passed to multiple
processes simultaneously. Also note that a different service may be
activated on incoming socket traffic than the one which is ultimately
configured to inherit the socket file descriptors. Or in other words: the
Service= setting of .socket units does not have to match the inverse of the
Sockets= setting of the .service it refers to.

    This option may appear more than once, in which case the list of socket
units is merged. If the empty string is assigned to this option, the list of
sockets is reset, and all prior uses of this setting will have no effect.
StartLimitInterval=, StartLimitBurst=

    Configure service start rate limiting. By default, services which are
started more than 5 times within 10 seconds are not permitted to start any
more times until the 10 second interval ends. With these two options, this
rate limiting may be modified. Use StartLimitInterval= to configure the
checking interval (defaults to DefaultStartLimitInterval= in manager
configuration file, set to 0 to disable any kind of rate limiting). Use
StartLimitBurst= to configure how many starts per interval are allowed
(defaults to DefaultStartLimitBurst= in manager configuration file). These
configuration options are particularly useful in conjunction with Restart=;
```

however, they apply to all kinds of starts (including manual), not just those triggered by the Restart= logic. Note that units which are configured for Restart= and which reach the start limit are not attempted to be restarted anymore; however, they may still be restarted manually at a later point, from which point on, the restart logic is again activated. Note that systemctl reset-failed will cause the restart rate counter for a service to be flushed, which is useful if the administrator wants to manually start a service and the start limit interferes with that.
StartLimitAction=

    Configure the action to take if the rate limit configured with StartLimitInterval= and StartLimitBurst= is hit. Takes one of none, reboot, reboot-force, reboot-immediate, poweroff, poweroff-force or poweroff-immediate. If none is set, hitting the rate limit will trigger no action besides that the start will not be permitted. reboot causes a reboot following the normal shutdown procedure (i.e. equivalent to systemctl reboot). reboot-force causes a forced reboot which will terminate all processes forcibly but should cause no dirty file systems on reboot (i.e. equivalent to systemctl reboot -f) and reboot-immediate causes immediate execution of the reboot(2) system call, which might result in data loss. Similar, poweroff, poweroff-force, poweroff-immediate have the effect of powering down the system with similar semantics. Defaults to none.
FailureAction=

    Configure the action to take when the service enters a failed state. Takes the same values as StartLimitAction= and executes the same actions. Defaults to none.
RebootArgument=

    Configure the optional argument for the reboot(2) system call if StartLimitAction= or FailureAction= is a reboot action. This works just like the optional argument to systemctl reboot command.
FileDescriptorStoreMax=

    Configure how many file descriptors may be stored in the service manager for the service using sd_pid_notify_with_fds(3)'s "FDSTORE=1" messages. This is useful for implementing service restart schemes where the state is serialized to /run and the file descriptors passed to the service manager, to allow restarts without losing state. Defaults to 0, i.e. no file descriptors may be stored in the service manager by default. All file descriptors passed to the service manager from a specific service are passed back to the service's main process on the next service restart. Any file descriptors passed to the service manager are automatically closed when POLLHUP or POLLERR is seen on them, or when the service is fully stopped and no job queued or being executed for it.
USBFunctionDescriptors=

    Configure the location of a file containing USB FunctionFS descriptors, for implementation of USB gadget functions. This is is used only in conjunction with a socket unit with ListenUSBFunction= configured. The contents of this file is written to the ep0 file after it is opened.

```
USBFunctionStrings=

    Configure the location of a file containing USB FunctionFS strings.
Behavior is similar to USBFunctionDescriptors= above.
```

Section [Install]

? Alias= :: A space-separated list of additional names this unit shall be installed under. The names listed here must have the same suffix (i.e. type) as the unit file name. This option may be specified more than once, in which case all listed names are used. At installation time, systemctl enable will create symlinks from these names to the unit filename. !!
? WantedBy=, RequiredBy= :: This option may be used more than once, or a space-separated list of unit names may be given. A symbolic link is created in the .wants/ or .requires/ directory of each of the listed units when this unit is installed by systemctl enable. This has the effect that a dependency of type Wants= or Requires= is added from the listed unit to the current unit. The primary result is that the current unit will be started when the listed unit is started. See the description of Wants= and Requires= in the [Unit] section for details. :: WantedBy=foo.service in a service bar.service is mostly equivalent to Alias=foo.service.wants/bar.service in the same file. In case of template units, systemctl enable must be called with an instance name, and this instance will be added to the .wants/ or .requires/ list of the listed unit. E.g. WantedBy=getty.target in a service getty@.service will result in systemctl enable getty@tty2.service creating a getty.target.wants/getty@tty2.service link to getty@.service. !!
? Also= :: Additional units to install/deinstall when this unit is installed/deinstalled. If the user requests installation/deinstallation of a unit with this option configured, systemctl enable and systemctl disable will automatically install/uninstall units listed in this option as well. :: This option may be used more than once, or a space-separated list of unit names may be given. !!
? DefaultInstance= :: In template unit files, this specifies for which instance the unit shall be enabled if the template is enabled without any explicitly set instance. This option has no effect in non-template unit files. The specified string must be usable as instance identifier. !!

Utilisation

Désinstallation

Voir aussi

- **(en)** http://app

Basé sur <Titre original de l'article> par <Auteur Original>.