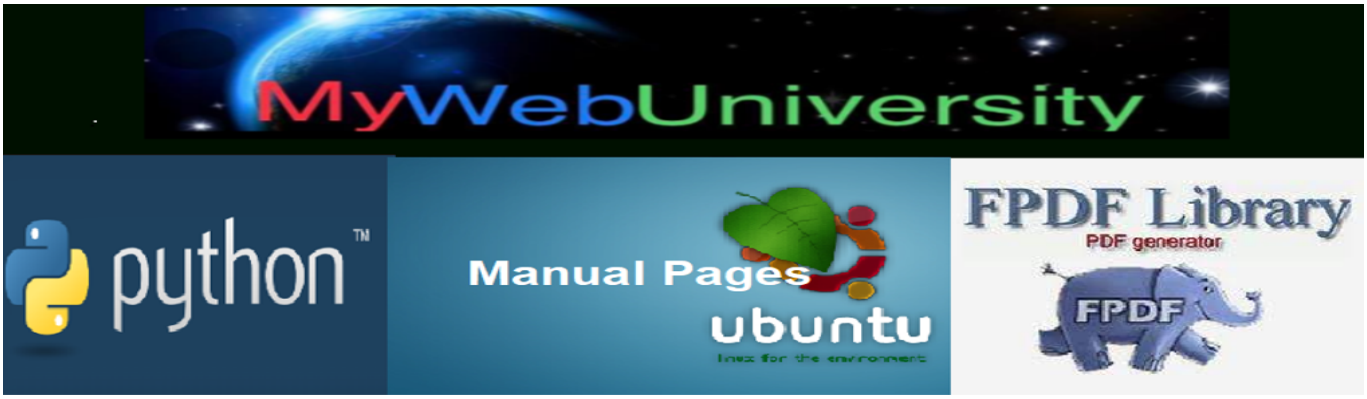

Full credit is given to the above companies including the Operating System (OS) that this PDF file was generated!

Rocky Enterprise Linux 9.2 Manual Pages on command 'systemd.environment-generator.7'

$ man systemd.environment-generator.7

SYSTEMD.ENVIRONMENT-GENERATOR(7) systemd.environment-generator

SYSTEMD.ENVIRONMENT-GENERATOR(7)

NAME

systemd.environment-generator - systemd environment file generators

SYNOPSIS

/lib/systemd/system-environment-generators/some-generator

/usr/lib/systemd/user-environment-generators/some-generator

/run/systemd/system-environment-generators/*

/etc/systemd/system-environment-generators/*

/usr/local/lib/systemd/system-environment-generators/*

/lib/systemd/system-environment-generators/*

/run/systemd/user-environment-generators/*

/etc/systemd/user-environment-generators/*

/usr/local/lib/systemd/user-environment-generators/*

/usr/lib/systemd/user-environment-generators/*

DESCRIPTION

Generators are small executables that live in /lib/systemd/system-environment-generators/ and other directories listed above. systemd(1) will execute those binaries very early at the startup of each manager and at configuration reload time, before running the generators described in systemd.generator(7) and before starting any units. Environment generators can override the environment that the manager exports to services and other processes.

Generators are loaded from a set of paths determined during compilation, as listed above.

System and user environment generators are loaded from directories with names ending in system-environment-generators/ and user-environment-generators/, respectively. Generators found in directories listed earlier override the ones with the same name in directories lower in the list. A symlink to /dev/null or an empty file can be used to mask a generator, thereby preventing it from running. Please note that the order of the two directories with the highest priority is reversed with respect to the unit load path, and generators in /run/ overwrite those in /etc/.

After installing new generators or updating the configuration, systemctl daemon-reload may be executed. This will re-run all generators, updating environment configuration. It will be used for any services that are started subsequently.

Environment file generators are executed similarly to unit file generators described in systemd.generator(7), with the following differences:

? Generators are executed sequentially in the alphanumerical order of the final component of their name. The output of each generator output is immediately parsed and used to update the environment for generators that run after that. Thus, later generators can use and/or modify the output of earlier generators.

? Generators are run by every manager instance, their output can be different for each user.

It is recommended to use numerical prefixes for generator names to simplify ordering.

EXAMPLES

Example 1. A simple generator that extends an environment variable if a directory exists in the file system

```
# 50-xdg-data-dirs.sh
#!/bin/bash
# set the default value
XDG_DATA_DIRS="${XDG_DATA_DIRS:-/usr/local/share/:/usr/share}"
# add a directory if it exists
if [[ -d /opt/foo/share ]]; then
    XDG_DATA_DIRS="/opt/foo/share:${XDG_DATA_DIRS}"
fi
# write our output
echo "XDG_DATA_DIRS=${XDG_DATA_DIRS}"
```

Example 2. A more complicated generator which reads existing configuration and mutates one

variable

```
# 90-rearrange-path.py
#!/usr/bin/env python3
"""
Proof-of-concept systemd environment generator that makes sure that bin dirs
are always after matching sbin dirs in the path.
(Changes /sbin:/bin:/foo/bar to /bin:/sbin:/foo/bar.)
This generator shows how to override the configuration possibly created by
earlier generators. It would be easier to write in bash, but let's have it
in Python just to prove that we can, and to serve as a template for more
interesting generators.
"""
import os
import pathlib
def rearrange_bin_sbin(path):
    """Make sure any pair of .../bin, .../sbin directories is in this order
    >>> rearrange_bin_sbin('/bin:/sbin:/usr/sbin:/usr/bin')
    '/bin:/sbin:/usr/bin:/usr/sbin'
    """
    items = [pathlib.Path(p) for p in path.split(':')]
    for i in range(len(items)):
        if 'sbin' in items[i].parts:
            ind = items[i].parts.index('sbin')
            bin = pathlib.Path(*items[i].parts[:ind], 'bin', *items[i].parts[ind+1:])
            if bin in items[i+1:]:
                j = i + 1 + items[i+1:].index(bin)
                items[i], items[j] = items[j], items[i]
    return ':'.join(p.as_posix() for p in items)
if __name__ == '__main__':
    path = os.environ['PATH'] # This should be always set.
                              # If it's not, we'll just crash, which is OK too.
    new = rearrange_bin_sbin(path)
    if new != path:
```

```
        print('PATH={}'.format(new))
```

Example 3. Debugging a generator

```
    SYSTEMD_LOG_LEVEL=debug VAR_A=something VAR_B="something else" \
    /lib/systemd/system-environment-generators/path-to-generator
```

SEE ALSO

systemd-environment-d-generator(8), systemd.generator(7), systemd(1), systemctl(1)