# SoS Documentation

*Release 4.2*

**Bryn Reeves**

**Aug 23, 2021**

# Contents

Sos is an extensible, portable, support data collection tool primarily aimed at Linux distributions and other UNIX-like operating systems.

This is the SoS developer documentation, for user documentation refer to:

https://github.com/sosreport/sos/wiki

This project is hosted at:

https://github.com/sosreport/sos

For the latest version, to contribute, and for more information, please visit the project pages or join the mailing list.

To clone the current main (development) branch run:

```
git clone git://github.com/sosreport/sos.git
```

CHAPTER 1

# Reporting bugs

Please report bugs via the mailing list or by opening an issue in the GitHub Issue Tracker

CHAPTER 2

# Mailing list

sos-devel is the mailing list for any sos-related questions and discussion. Patch submissions and reviews are welcome too.

CHAPTER 3

# Patches and pull requests

Patches can be submitted via the mailing list or as GitHub pull requests. If using GitHub please make sure your branch applies to the current main branch as a 'fast forward' merge (i.e. without creating a merge commit). Use the git rebase command to update your branch to the current main branch if necessary.

CHAPTER 4

# Documentation

User and API documentation is automatically generated using Sphinx and Read the Docs.

## 4.1 Wiki

How to write a plugin

How to write a policy

Plugin options

To help get your changes merged quickly with as few revisions as possible please refer to the Contributor Guidelines when submitting patches or pull requests.

CHAPTER 5

# Installation

## 5.1 Manual Installation

```
python3 setup.py install
```

## 5.2 Pre-built Packaging

Fedora/RHEL users install via yum:

```
yum install sos
```

Debian users install via apt:

```
apt install sosreport
```

Ubuntu (14.04 LTS and above) users install via apt:

```
sudo apt install sosreport
```

CHAPTER 6

# API

## 6.1 Core Reference

### 6.1.1 `sos.archive` — Archive Interface

**class** sos.archive.**Archive**
Bases: `object`

Abstract base class for archives.

**add_binary**(*content*, *dest*)

**add_dir**(*path*)

**add_file**(*src*, *dest=None*)

**add_link**(*source*, *link_name*)

**add_node**(*path*, *mode*, *device*)

**add_string**(*content*, *dest*, *mode='w'*)

**classmethod archive_type**()
Returns the archive class's name as a string.

**cleanup**()
Clean up any temporary resources used by an Archive class.

**finalize**(*method*)
Finalize an archive object via method. This may involve creating An archive that is subsequently compressed or simply closing an archive that supports in-line handling. If method is automatic then the following methods are tried in order: xz, gzip

**get_archive_path**()
Return a string representing the path to the temporary archive. For archive classes that implement in-line handling this will be the archive file itself. Archives that use a directory based cache prior to packaging should return the path to the temporary directory where the report content is located

**get_tmp_dir**()
Return a temporary directory that clients of the archive may use to write content to. The content of the path is guaranteed to be included in the generated archive.

**log = <Logger sos (WARNING)>**

**log_debug**(*msg*)

**log_error**(*msg*)

**log_info**(*msg*)

**log_warn**(*msg*)

**name_max**()
Return the maximum file name length this archive can support. This is the lesser of the name length limit of the archive format and any temporary file system based cache.

**set_debug**(*debug*)

**class** sos.archive.**FileCacheArchive**(*name*, *tmpdir*, *policy*, *threads*, *enc_opts*, *sysroot*, *manifest=None*)
Bases: sos.archive.Archive

Abstract superclass for archive types that use a temporary cache directory in the file system.

**add_binary**(*content*, *dest*)

**add_dir**(*path*)
Create a directory in the archive.

**Parameters** **path** – the path in the host file system to add

**add_file**(*src*, *dest=None*)

**add_final_manifest_data**(*method*)
Adds component-agnostic data to the manifest so that individual SoSComponents do not need to redundantly add these manually

**add_link**(*source*, *link_name*)

**add_node**(*path*, *mode*, *device*)

**add_string**(*content*, *dest*, *mode='w'*)

**cleanup**()
Clean up any temporary resources used by an Archive class.

**dest_path**(*name*)

**finalize**(*method*)
Finalize an archive object via method. This may involve creating An archive that is subsequently compressed or simply closing an archive that supports in-line handling. If method is automatic then the following methods are tried in order: xz, gzip

**get_archive_path**()
Return a string representing the path to the temporary archive. For archive classes that implement in-line handling this will be the archive file itself. Archives that use a directory based cache prior to packaging should return the path to the temporary directory where the report content is located

**get_tmp_dir**()
Return a temporary directory that clients of the archive may use to write content to. The content of the path is guaranteed to be included in the generated archive.

**join_sysroot**(*path*)

**makedirs**(*path*, *mode=448*)
Create path, including leading components.

Used by sos.sosreport to set up sos_* directories.

**name_max**()
Return the maximum file name length this archive can support. This is the lesser of the name length limit of the archive format and any temporary file system based cache.

**open_file**(*path*)

**rename_archive_root**(*cleaner*)
Rename the archive to an obfuscated version using an initialized SoSCleaner instance

**class** sos.archive.**TarFileArchive**(*name*, *tmpdir*, *policy*, *threads*, *enc_opts*, *sysroot*, *manifest=None*)
Bases: sos.archive.FileCacheArchive

archive class using python TarFile to create tar archives

**copy_permissions_filter**(*tarinfo*)

**get_selinux_context**(*path*)

**method = None**

**name**()

**name_max**()
Return the maximum file name length this archive can support. This is the lesser of the name length limit of the archive format and any temporary file system based cache.

**set_tarinfo_from_stat**(*tar_info*, *fstat*, *mode=None*)

### 6.1.2 sos.collector.clusters — Cluster Interface

**class** sos.collector.clusters.**Cluster**(*commons*)
Bases: object

This is the class that cluster profiles should subclass in order to add support for different clustering technologies and environments to sos-collector.

A profile should at minimum define a package that indicates the node is configured for the type of cluster the profile is intended to serve and then additionally be able to return a list of enumerated nodes via the get_nodes() method

**Parameters** **commons** (dict) – The commons dict containing system information. The same as what is handed to Plugin()

**Variables**

- **option_list** (list of tuples) – Options supported by the profile, and set by the –cluster-option cmdline arg
- **packages** (tuple) – What package(s) should this profile enable on
- **sos_plugins** (list) – Which plugins to forcibly enable for node reports
- **sos_plugin_options** (dict) – Plugin options to forcibly set for nodes
- **sos_preset** (str) – A SoSReport preset to forcibly enable on nodes
- **cluster_name** (str) – The name of the cluster type

**add_default_ssh_key**(*key*)
Some clusters generate and/or deploy well-known and consistent SSH keys across environments. If this is the case, the cluster profile may call this command so that subsequent node connections will use that key rather than prompting the user for one or a password.

Note this will only function if collector is being run locally on the primary node.

**check_enabled**()
This may be overridden by clusters

This is called by sos collect on each cluster type that exists, and is meant to return True when the cluster type matches a criteria that indicates that is the cluster type is in use.

Only the first cluster type to determine a match is run

**Returns** `True` if the cluster profile should be used, or `False`

**Return type** `bool`

**check_node_is_primary**(*node*)
In the event there are multiple primaries, or if the collect command is being run from a system that is technically capable of enumerating nodes but the cluster profiles needs to specify primary-specific options for other nodes, override this method in the cluster profile

**Parameters** **node** (`SoSNode`) – The node for the cluster to check

**exec_primary_cmd**(*cmd*, *need_root=False*)
Used to retrieve command output from a (primary) node in a cluster

**Parameters**

- **cmd** (`str`) – The command to run
- **need_root** (`bool`) – Does the command require root privileges

**Returns** The output and status of *cmd*

**Return type** `dict`

**format_node_list**()
Format the returned list of nodes from a cluster into a known format. This being a list that contains no duplicates

**Returns** A list of nodes, without extraneous entries from cmd output

**Return type** `list`

**get_node_label**(*node*)
Used by `SosNode()` to retrieve the appropriate label from the cluster as set by `set_node_label()` in the cluster profile.

**Parameters** **node** (`str`) – The name of the node to get a label for

**Returns** The label to use for the node's report

**Return type** `str`

**get_nodes**()
This MUST be overridden by a cluster profile subclassing this class

A cluster should use this method to return a list or string that contains all the nodes that a report should be collected from

**Returns** A list of node FQDNs or IP addresses

**Return type** `list` or `None`

**get_option**(*option*)
This is used to by clusters to check if a cluster option was supplied to sos collect

**Parameters** **option** (str) – The name of the option to fetch

**Returns** The value of the requested option if it exists, or False

**log_debug**(*msg*)
Used to print debug messages

**log_error**(*msg*)
Used to print error messages

**log_info**(*msg*)
Used to print info messages

**log_warn**(*msg*)
Used to print warning messages

**classmethod name**()
Returns the cluster's name as a string.

**set_node_label**(*node*)
This may be overridden by clusters profiles subclassing this class

If there is a distinction between primaries and nodes, or types of nodes, then this can be used to label the sosreport archive differently

**set_node_options**(*node*)
If there is a need to set specific options on ONLY the non-primary nodes in a collection, override this method in the cluster profile and do that here.

**Parameters** **node** (SoSNode) – The non-primary node

**set_primary_options**(*node*)
If there is a need to set specific options in the sos command being run on the cluster's primary nodes, override this method in the cluster profile and do that here.

**Parameters** **node** (SoSNode) – The primary node

**setup**()
This MAY be used by a cluster to do prep work in case there are extra commands to be run even if a node list is given by the user, and thus get_nodes() would not be called

### 6.1.3 sos.cleaner.parsers — Parser Interface

**class** sos.cleaner.parsers.**SoSCleanerParser**(*config={}*)
Bases: object

Parsers are used to build objects that will take a line as input, parse it for a particular pattern (E.G. IP addresses) and then make any necessary subtitutions by referencing the SoSMap() associated with the parser.

Ideally a new parser subclass will only need to set the class level attrs in order to be fully functional.

**Parameters** **conf_file** (str) – The configuration file to read from

**Variables**

- **name** (str) – The parser name, used in logging errors
- **regex_patterns** (list) – A list of regex patterns to iterate over for every line processed
- **mapping** (SoSMap()) – Used by the parser to store and obfuscate matches

- **map_file_key** (str) – The key in the map_file to read when loading previous obfuscation matches
- **prep_map_file** – File to read from an archive to pre-seed the map with matches. E.G. ip_addr for loading IP addresses

**get_map_contents**()
Get the contents of the mapping used by the parser

**Returns** All matches and their obfuscate counterparts

**Return type** dict

**parse_line**(*line*)
This will be called for every line in every file we process, so that every parser has a chance to scrub everything.

**Parameters** **line** (str) – The line to parse for possible matches for obfuscation

**Returns** The obfsucated line, and the number of changes made

**Return type** tuple, (str, int))

**parse_string_for_keys**(*string_data*)
Parse a given string for instances of any obfuscated items, without applying the normal regex comparisons first. This is mainly used to obfuscate filenames that have, for example, hostnames in them.

Rather than try to regex match the string_data, just use the builtin checks for substrings matching known obfuscated keys

**Parameters** **string_data** (str) – The line to be parsed

**Returns** The obfuscated line

**Return type** str

### 6.1.4 sos.policies — Policy Interface

**class** sos.policies.**GenericPolicy**(*sysroot=None*, *probe_runtime=True*)
Bases: sos.policies.Policy

This Policy will be returned if no other policy can be loaded. This should allow for IndependentPlugins to be executed on any system

**get_msg**()
This method is used to prepare the preamble text to display to the user in non-batch mode. If your policy sets self.distro that text will be substituted accordingly. You can also override this method to do something more complicated.

**Returns** Formatted banner message string

**Return type** str

**class** sos.policies.**Policy**(*sysroot=None*, *probe_runtime=True*)
Bases: object

Policies represent distributions that sos supports, and define the way in which sos behaves on those distributions. A policy should define at minimum a way to identify the distribution, and a package manager to allow for package based plugin enablement.

Policies also control preferred ContainerRuntime()'s, upload support to default locations for distribution vendors, disclaimer text, and default presets supported by that distribution or vendor's products.

Every Policy will also need at least one "tagging class" for plugins.

**Parameters**

- **sysroot** (str or None) – Set the sysroot for the system, if not /
- **probe_runtime** (bool) – Should the Policy try to load a ContainerRuntime

**Variables**

- **distro** (str) – The name of the distribution the Policy represents
- **vendor** (str) – The name of the vendor producing the distribution
- **vendor_urls** (list of tuples formatted (``description, url)'') – List of URLs for the vendor's website, or support portal
- **vendor_text** (str) – Additional text to add to the banner message
- **name_pattern** (str) – The naming pattern to be used for naming archives generated by sos. Values of *legacy*, and *friendly* are preset patterns. May also be set to an explicit custom pattern, see *get_archive_name()*

**add_preset**(*name=None*, *desc=None*, *note=None*, *opts=SoSOptions()*)
Add a new on-disk preset and write it to the configured presets path.

**Parameters** **preset** – the new PresetDefaults to add

**check**(*remote=''*)
This function is responsible for determining if the underlying system is supported by this policy.

If *remote* is provided, it should be the contents of os-release from a remote host, or a similar vendor-specific file that can be used in place of a locally available file.

**Returns** True if the Policy should be loaded, else False

**Return type** bool

**display_results**(*archive*, *directory*, *checksum*, *archivestat=None*, *map_file=None*)
Display final information about a generated archive

**Parameters**

- **archive** (str) – The name of the archive that was generated
- **directory** (str) – The build directory for sos if –build was used
- **checksum** (str) – The checksum of the archive
- **archivestat** (*os.stat_result*) – stat() information for the archive
- **map_file** (str) – If sos clean was invoked, the location of the mapping file for this run

**dist_version**()
Return the OS version

**find_preset**(*preset*)
Find a preset profile matching the specified preset string.

**Parameters** **preset** – a string containing a preset profile name.

**Returns** a matching PresetProfile.

**forbidden_paths**
This property is used to determine the list of forbidden paths set by the policy. Note that this property will construct a *cumulative* list based on all subclasses of a given policy.

**Returns** All patterns of policy forbidden paths

**Return type** list

**get_archive_name**()
This function should return the filename of the archive without the extension.

This uses the policy's *name_pattern* attribute to determine the name. There are two pre-defined naming patterns - *legacy* and *friendly* that give names like the following:

- legacy - *sosreport-tux.123456-20171224185433*
- friendly - *sosreport-tux-mylabel-123456-2017-12-24-ezcfcop.tar.xz*

A custom name_pattern can be used by a policy provided that it defines name_pattern using a format() style string substitution.

Usable substitutions are:

- name - the short hostname of the system
- label - the label given by –label
- case - the case id given by –case-id or –ticker-number
- rand - a random string of 7 alpha characters

Note that if a datestamp is needed, the substring should be set in *name_pattern* in the format accepted by `strftime()`.

**Returns** A name to be used for the archive, as expanded from the Policy *name_pattern*

**Return type** `str`

**get_msg**()
This method is used to prepare the preamble text to display to the user in non-batch mode. If your policy sets self.distro that text will be substituted accordingly. You can also override this method to do something more complicated.

**Returns** Formatted banner message string

**Return type** `str`

**get_preferred_archive**()
Return the class object of the prefered archive format for this platform

**get_preferred_hash_name**()
Returns the string name of the hashlib-supported checksum algorithm to use

**host_sysroot**()
Get the host's default sysroot

**Returns** Host sysroot

**Return type** `str` or `None`

**in_container**()
Are we running inside a container?

**Returns** `True` if in a container, else `False`

**Return type** `bool`

**is_root**()
This method should return true if the user calling the script is considered to be a superuser

**Returns** `True` if user is superuser, else `False`

**Return type** `bool`

**load_presets**(*presets_path=None*)
Load presets from disk.

Read JSON formatted preset data from the specified path, or the default location at `/var/lib/sos/presets`.

**Parameters** **presets_path** – a directory containing JSON presets.

**match_plugin**(*plugin_classes*)
Determine what subclass of a Plugin should be used based on the tagging classes assigned to the Plugin

**Parameters** **plugin_classes** (`list`) – The classes that the Plugin subclasses

**Returns** The first subclass that matches one of the Policy's *valid_subclasses*

**Return type** A tagging class for Plugins

**pkg_by_name**(*pkg*)
Wrapper to retrieve a package from the Policy's package manager

**Parameters** **pkg** (`str`) – The name of the package

**Returns** The first package that matches *pkg*

**Return type** `str`

**post_work**()
This function is called after the sosreport has been generated.

**pre_work**()
This function is called prior to collection.

**probe_preset**()
Return a `PresetDefaults` object matching the runing host.

Stub method to be implemented by derived policy classes.

**Returns** a `PresetDefaults` object.

**register_presets**(*presets*, *replace=False*)
Add new presets to this policy object.

Merges the presets dictionary `presets` into this `Policy` object, or replaces the current presets if `replace` is `True`.

`presets` should be a dictionary mapping `str` preset names to `<class PresetDefaults>` objects specifying the command line defaults.

**Parameters**

- **presets** – dictionary of presets to add or replace
- **replace** – replace presets rather than merge new presets.

**set_commons**(*commons*)
Set common host data for the Policy to reference

**classmethod set_forbidden_paths**()
Use this to *append* policy-specifc forbidden paths that apply to all plugins. Setting this classmethod on an invidual policy will *not* override subclass-specific paths

**validate_plugin**(*plugin_class*, *experimental=False*)
Verifies that the plugin_class should execute under this policy

**Parameters** **plugin_class** (*`A Plugin() tagging class`*) – The tagging class being checked

**Returns** `True` if the *plugin_class* is allowed by the policy

**Return type** `bool`

### 6.1.5 `sos.report.plugins` — Plugin Interface

This exports methods available for use by plugins for sos

**class** `sos.report.plugins.`**CosPlugin**
Bases: `object`

Tagging class for Container-Optimized OS

**class** `sos.report.plugins.`**DebianPlugin**
Bases: `object`

Tagging class for Debian Linux

**class** `sos.report.plugins.`**ExperimentalPlugin**
Bases: `object`

Tagging class that indicates that this plugin is experimental

**class** `sos.report.plugins.`**IndependentPlugin**
Bases: `object`

Tagging class for plugins that can run on any platform

**class** `sos.report.plugins.`**Plugin**(*commons*)
Bases: `object`

This is the base class for sosreport plugins. Plugins should subclass this and set the class variables where applicable.

**Parameters** **commons** (`dict`) – A set of information that is shared internally so that plugins may access the same dataset. This is provided automatically by sos

Each *Plugin()* subclass should also subclass at least one tagging class, e.g. `RedHatPlugin`, to support that distribution. If different distributions require different collections, each distribution should have its own subclass of the Plugin that also subclasses the tagging class for their respective distributions.

**Variables**

- **plugin_name** (`str`) – The name of the plugin, will be returned by *name()*
- **packages** (`tuple`) – Package name(s) that, if installed, enable this plugin
- **files** (`tuple`) – File path(s) that, if present, enable this plugin
- **commands** (`tuple`) – Executables that, if present, enable this plugin
- **kernel_mods** (`tuple`) – Kernel module(s) that, if loaded, enable this plugin
- **services** (`tuple`) – Service name(s) that, if running, enable this plugin
- **architectures** (`tuple`, or `None`) – Architecture(s) this plugin is enabled for. Defaults to 'none' to enable on all arches.
- **profiles** (`tuple`) – Name(s) of profile(s) this plugin belongs to
- **plugin_timeout** (`int`) – Timeout in seconds for this plugin as a whole
- **cmd_timeout** (`int`) – Timeout in seconds for individual commands

**add_alert**(*alertstring*)
Add an alert to the collection of alerts for this plugin. These will be displayed in the report

**Parameters** **alertstring** (str) – The text to add as an alert

**add_blockdev_cmd**(*cmds*, *devices='block'*, *timeout=None*, *sizelimit=None*, *chroot=True*, *runat=None*, *env=None*, *binary=False*, *prepend_path=None*, *whitelist=[]*, *blacklist=[]*, *tags=[]*, *priority=10*)

Run a command or list of commands against storage-related devices.

Any commands specified by cmd will be iterated over the list of the specified devices. Commands passed to this should include a '%(dev)s' variable for substitution.

**Parameters**

- **cmds** (str or a list of strings) – The command(s) to run against the list of devices
- **devices** (str or a list of device paths) – The device paths to run *cmd* against. If set to *block* or *fibre*, the commands will be run against the matching list of discovered devices
- **timeout** (int) – Timeout in seconds to allow each *cmd* to run
- **sizelimit** (int) – Maximum amount of output to collect, in MB
- **chroot** (bool) – Should sos chroot the command(s) being run
- **runat** (str) – Set the filesystem location to execute the command from
- **env** (dict) – Set environment variables for the command(s) being run
- **binary** (bool) – Is the output collected going to be binary data
- **prepend_path** (str or None) – The leading path for block device names
- **whitelist** (list of str) – Limit the devices the *cmds* will be run against to devices matching these item(s)
- **blacklist** (list of str) – Do not run *cmds* against devices matching these item(s)

**add_cmd_output**(*cmds*, *suggest_filename=None*, *root_symlink=None*, *timeout=None*, *stderr=True*, *chroot=True*, *runat=None*, *env=None*, *binary=False*, *sizelimit=None*, *pred=None*, *subdir=None*, *changes=False*, *foreground=False*, *tags=[]*, *priority=10*, *cmd_as_tag=False*)

Run a program or a list of programs and collect the output

Output will be limited to *sizelimit*, collecting the last X amount of command output matching *sizelimit*. Unless *suggest_filename* is set, the file that the output is saved to will match the command as it was executed, and will be saved under *sos_commands/$plugin*

**Parameters**

- **cmds** (str or a list of strings) – The command(s) to execute
- **suggest_filename** (str) – Override the name of the file output is saved to within the archive
- **root_symlink** (str) – If set, create a symlink with this name in the archive root
- **timeout** (int) – Timeout in seconds to allow each *cmd* to run for
- **stderr** (bool) – Should stderr output be collected
- **chroot** (bool) – Should sos chroot the *cmds* being run
- **runat** (str) – Run the *cmds* from this location in the filesystem
- **env** (dict) – Set environment variables for the *cmds* being run
- **binary** (bool) – Is the command expected to produce binary output
- **sizelimit** (int) – Maximum amount of output in MB to save

- **pred** (SoSPredicate) – A predicate to gate if *cmds* should be collected or not
- **subdir** (str) – Save output to this subdirectory, within the plugin's directory under sos_commands
- **changes** (int) – Do *cmds* have the potential to change system state
- **foreground** (bool) – Should the *cmds* be run in the foreground, with an attached TTY
- **tags** (str or a list of strings) – A tag or set of tags to add to the metadata entries for the *cmds* being run
- **priority** (int) – The priority with which this command should be run, lower values will run before higher values
- **cmd_as_tag** (bool) – Should the command string be automatically formatted to a tag?

**add_cmd_tags**(*tagdict*)

Retroactively add tags to any commands that have been run by this plugin that match a given regex

**Parameters** **tagdict** (dict) – A dict containing the command regex and associated tags

*tagdict* takes the form of {cmd_regex: tags}, for example to tag all commands starting with *foo* with the tag *bar*, use {'foo.*': ['bar']}

**add_copy_spec**(*copyspecs*, *sizelimit=None*, *maxage=None*, *tailit=True*, *pred=None*, *tags=[]*)

Add a file, directory, or regex matching filepaths to the archive

**Parameters**

- **copyspecs** (str or a list of strings) – A file, directory, or regex matching filepaths
- **sizelimit** (int) – Limit the total size of collections from *copyspecs* to this size in MB
- **maxage** (int) – Collect files with *mtime* not older than this many hours
- **tailit** (bool) – Should a file that exceeds *sizelimit* be tail'ed to fit the remaining space to meet *sizelimit*
- **pred** (SoSPredicate) – A predicate to gate if *copyspecs* should be collected
- **tags** (str or a list of strings) – A tag or set of tags to add to the metadata information for this collection

*copyspecs* will be expanded and/or globbed as appropriate. Specifying a directory here will cause the plugin to attempt to collect the entire directory, recursively.

Note that *sizelimit* is applied to each *copyspec*, not each file individually. For example, a copyspec of `['/etc/foo', '/etc/bar.conf']` and a *sizelimit* of 25 means that sos will collect up to 25MB worth of files within */etc/foo*, and will collect the last 25MB of */etc/bar.conf*.

**add_custom_text**(*text*)

Append text to the custom text that is included in the report. This is freeform and can include html.

**Parameters** **text** (str) – The text to include in the report

**add_default_collections**()

Based on the class attrs defined for plugin enablement, add a standardized set of collections before we call the plugin's own setup() method.

**add_env_var**(*name*)

Add an environment variable to the list of to-be-collected env vars.

Collected environment variables will be saved to an *environment* file in the archive root, and any variable specified for collection will be collected in lowercase, uppercase, and the form provided

**Parameters** **name** (str) – The name of the environment variable to collect

**add_file_tags**(*tagdict*)
Apply a tag to a file matching a given regex, for use when a file is copied by a more generic copyspec.

**Parameters** **tagdict** (dict) – A dict containing the filepatterns to match and the tag(s) to apply to those files

*tagdict* takes the form *{file_pattern: tag}*, E.G. to match all bond devices from /proc/net/bonding with the tag *bond*, use *{'/proc/net/bonding/bond.*': ['bond']}*

**add_forbidden_path**(*forbidden*, *recursive=False*)
Specify a path, or list of paths, to not copy, even if it's part of an add_copy_spec() call

**Parameters**

- **forbidden** (str or a list of strings) – A filepath to forbid collection from
- **recursive** – Should forbidden glob be applied recursively

**add_journal**(*units=None*, *boot=None*, *since=None*, *until=None*, *lines=None*, *allfields=False*, *output=None*, *timeout=None*, *identifier=None*, *catalog=None*, *sizelimit=None*, *pred=None*, *tags=None*, *priority=10*)
Collect journald logs from one of more units.

**Parameters**

- **units** (str or a list of strings) – Which journald units to collect
- **boot** (str) – A boot index using the journalctl syntax. The special values 'this' and 'last' are also accepted.
- **since** (str) – Start time for journal messages
- **until** (str) – End time forjournal messages
- **lines** (int) – The maximum number of lines to be collected
- **allfields** (bool) – Include all journal fields regardless of size or non-printable characters
- **output** (str) – Journalctl output control string, for example "verbose"
- **timeout** (int) – An optional timeout in seconds
- **identifier** (str) – An optional message identifier
- **catalog** (bool) – Augment lines with descriptions from the system catalog
- **sizelimit** (int) – Limit to the size of output returned in MB. Defaults to the value of –log-size.

**add_service_status**(*services*, ***kwargs*)
Collect service status information based on the InitSystem used

**Parameters**

- **services** (str or a list of strings) – Service name(s) to collect statuses for
- **kwargs** – Optional arguments to pass to _add_cmd_output (timeout, predicate, suggest_filename,..)

**add_string_as_file**(*content*, *filename*, *pred=None*)
Add a string to the archive as a file

**Parameters**

- **content** (str) – The string to write to the archive

- **filename** (str) – The name of the file to write *content* to
- **pred** (SoSPredicate) – A predicate to gate if the string should be added to the archive or not

**check_enabled**()
This method will be used to verify that a plugin should execute given the condition of the underlying environment.

The default implementation will return True if none of class.files, class.packages, nor class.commands is specified. If any of these is specified the plugin will check for the existence of any of the corresponding paths, packages or commands and return True if any are present.

For SCLPlugin subclasses, it will check whether the plugin can be run for any of installed SCLs. If so, it will store names of these SCLs on the plugin class in addition to returning True.

For plugins with more complex enablement checks this method may be overridden.

**Returns** True if the plugin should be run for this system, else False

**Return type** bool

**check_is_architecture**()
Checks whether or not the system is running on an architecture that the plugin allows. If not architecture is set, assume plugin can run on all arches.

**Returns** True if the host's architecture allows the plugin to run, else False

**Return type** bool

**check_process_by_name**(*process*)
Checks if a named process is found in /proc/[0-9]*/cmdline.

**Parameters** **process** (str) – The name of the process

**Returns** True if the process exists, else False

**Return type** bool

**check_timeout**()
Checks to see if the plugin has hit its timeout.

This is set when the sos.collect_plugin() method hits a timeout and terminates the thread. From there, a Popen() call can still continue to run, and we need to manually terminate it. Thus, check_timeout() should only be called in sos_get_command_output().

Since sos_get_command_output() is not plugin aware, this method is handed to that call to use as a polling method, to avoid passing the entire plugin object.

**Returns** True if timeout has been hit, else False

**Return type** bool

**cmdtimeout**
Returns either the default command timeout value, the value as provided on the commandline via -k plugin.cmd-timeout=value, or the value of the global –cmd-timeout option.

**collect**()
Collect the data for a plugin.

**collect_cmd_output**(*cmd*, *suggest_filename=None*, *root_symlink=False*, *timeout=None*, *stderr=True*, *chroot=True*, *runat=None*, *env=None*, *binary=False*, *sizelimit=None*, *pred=None*, *subdir=None*, *tags=[]*)
Execute a command and save the output to a file for inclusion in the report, then return the results for further use by the plugin

**Parameters**

- **cmd** (str) – The command to run
- **suggest_filename** – Filename to use when writing to the archive
- **suggest_filename** – str
- **root_symlink** (bool) – Create a symlink in the archive root
- **timeout** (int) – Time in seconds to allow a cmd to run
- **stderr** (bool) – Write stderr to stdout?
- **chroot** (bool) – Perform chroot before running cmd?
- **runat** (str) – Run the command from this location, overriding chroot
- **env** (dict) – Environment vars to set for the cmd
- **binary** (bool) – Is the output in binary?
- **sizelimit** (int) – Maximum size in MB of output to save
- **subdir** (str) – Subdir in plugin directory to save to
- **changes** (bool) – Does this cmd potentially make a change on the system?
- **tags** (str or a list of strings) – Add tags in the archive manifest

**Returns** *cmd* exit status, output, and the filepath within the archive output was saved to

**Return type** dict

**container_exists**(*name*)

If a container runtime is present, check to see if a container with a given name is currently running

**Parameters** **name** (str) – The name of the container to check presence of

**Returns** True if *name* exists, else False

**Return type** bool

**default_enabled**()

This decides whether a plugin should be automatically loaded or only if manually specified in the command line.

**do_cmd_output_sub**(*cmd*, *regexp*, *subst*)

Apply a regexp substitution to command output archived by sosreport.

This is used to obfuscate sensitive information captured by command output collection via plugins.

**Parameters**

- **cmd** (str) – The command name/binary name for collected output that needs to be obfuscated. Internally globbed with a leading and trailing *
- **regexp** (str or compile re object) – A regex to match the contents of the command output against
- **subst** (str) – The substitution string used to replace matches from *regexp*

**Returns** Number of replacements made

**Return type** int

**do_cmd_private_sub**(*cmd*, *desc=''*)

Remove certificate and key output archived by sos report. Any matching instances are replaced with: '——SCRUBBED' and this function does not take a regexp or substituting string.

**Parameters**

- **cmd** (str) – The name of the binary to scrub certificate output from
- **desc** (str) – An identifier to add to the *SCRUBBED* header line

**Returns** Number of replacements made

**Return type** int

**do_file_private_sub**(*pathregex*, *desc=''*)
Scrub certificate/key/etc information from files collected by sos.

Files matching the provided pathregex are searched for content that resembles certificate, ssh keys, or similar information. Any matches are replaced with "—–SCRUBBED $desc" where *desc* is a description of the specific type of content being replaced, e.g. "—–SCRUBBED RSA PRIVATE KEY" so that support representatives can at least be informed of what type of content it was originally.

**Parameters**

- **pathregex** (str) – A string or regex of a filename to match against
- **desc** (str) – A description of the replaced content

**do_file_sub**(*srcpath*, *regexp*, *subst*)
Apply a regexp substitution to a file archived by sosreport.

**Parameters**

- **srcpath** (str) – Path in the archive where the file can be found
- **regexp** (str or compiled re object) – A regex to match the contents of the file
- **subst** (str) – The substitution string to be used to replace matches within the file

**Returns** Number of replacements made

**Return type** int

**do_path_regex_sub**(*pathexp*, *regexp*, *subst*)
Apply a regexp substituation to a set of files archived by sos. The set of files to be substituted is generated by matching collected file pathnames against *pathexp*.

**Parameters**

- **pathexp** (str or compiled re object) – A regex to match filenames within the archive
- **regexp** (str or compiled re object) – A regex to match against the contents of each file
- **subst** (str) – The substituion string to be used to replace matches

**exec_cmd**(*cmd*, *timeout=None*, *stderr=True*, *chroot=True*, *runat=None*, *env=None*, *binary=False*, *pred=None*, *foreground=False*, *container=False*, *quotecmd=False*)
Execute a command right now and return the output and status, but do not save the output within the archive.

Use this method in a plugin's setup() if command output is needed to build subsequent commands added to a report via add_cmd_output().

**Parameters**

- **cmd** (str) – The command to run
- **timeout** (int) – Time in seconds to allow a cmd to run
- **stderr** (bool) – Write stderr to stdout?
- **chroot** (bool) – Perform chroot before running cmd?

- **runat** (str) – Run the command from this location, overriding chroot
- **env** (dict) – Environment vars to set for the cmd
- **binary** (bool) – Is the output in binary?
- **pred** (SoSPredicate) – A predicate to gate execution of the *cmd*
- **foreground** (bool) – Run the *cmd* in the foreground with a TTY
- **container** (str) – Execute this command in a container with this name
- **quotecmd** (bool) – Whether the cmd should be quoted.

**Returns** Command exit status and output

**Return type** dict

**file_grep**(*regexp*, **fnames*)
Grep through file(s) for a specific string or regex

**Parameters**

- **regexp** (str) – The string or regex to search for
- **fnames** (str, list of string, or open file objects) – Paths to grep through

**Returns** Lines matching *regexp*

**Return type** str

**filter_namespaces**(*ns_list*, *ns_pattern=None*, *ns_max=None*)
Filter a list of namespaces by regex pattern or max number of namespaces (options originally present in the networking plugin.)

**fmt_container_cmd**(*container*, *cmd*, *quotecmd=False*)
Format a command to be executed by the loaded ContainerRuntime in a specified container

**Parameters**

- **container** (str) – The name of the container to execute the *cmd* in
- **cmd** (str) – The command to run within the container
- **quotecmd** (bool) – Whether the cmd should be quoted.

**Returns** The command to execute so that the specified *cmd* will run within the *container* and not on the host

**Return type** str

**generate_copyspec_tags**()
After file collections have completed, retroactively generate manifest entries to apply tags to files copied by generic copyspecs

**get_all_containers_by_regex**(*regex*, *get_all=False*)
Get a list of all container names and ID matching a regex

**Parameters**

- **regex** (str) – The regular expression to match
- **get_all** (bool) – Return all containers found, even terminated ones

**Returns** All container IDs and names matching regex

**Return type** list of tuples as (id, name)

**get_all_options**()
return a list of all options selected

**get_cmd_output_path**(*name=None*, *make=True*)
Get the path where this plugin will save command output

**Parameters**

- **name** (str or None) – Optionally specify a filename to use as part of the command output path
- **make** (bool) – Attempt to create the command output path

**Returns** The path where the plugin will write command output data within the archive

**Return type** str

**get_container_by_name**(*name*)
Get the container ID for a specific container

**Parameters** **name** (str) – The name of the container

**Returns** The ID of the container if it exists

**Return type** str or None

**get_container_images**(*runtime=None*)
Return a list of all image names from the Policy's ContainerRuntime

If *runtime* is not provided, use the Policy default. If the specified *runtime* is not loaded, return empty.

**Parameters** **runtime** (str) – The container runtime to use, if not using the default runtime detected by the Policy

**Returns** A list of container images known to the *runtime*

**Return type** list

**get_container_logs**(*container*, ***kwargs*)
Helper to get the logs output for a given container

Supports passthru of add_cmd_output() options

**Parameters**

- **container** (str) – The name of the container to retrieve logs from
- **kwargs** – Any kwargs supported by add_cmd_output() are supported here

**get_container_volumes**(*runtime=None*)
Return a list of all volume names from the Policy's ContainerRuntime

If *runtime* is not provided, use the Policy default. If the specified *runtime* is not loaded, return empty.

**Parameters** **runtime** (str) – The container runtime to use, if not using the default runtime detected by the Policy

**Returns** A list of container volumes known to the *runtime*

**Return type** list

**get_containers**(*runtime=None*, *get_all=False*)
Return a list of all container IDs from the Policy ContainerRuntime

If *runtime* is not provided, use the Policy default

**Parameters**

- **runtime** (str) – The container runtime to use, if not the default runtime detected and loaded by the Policy
- **get_all** (bool) – Return all containers known to the *runtime*, even those that have terminated

**Returns** All container IDs found by the ContainerRuntime

**Return type** list

**get_description**()
This function will return the description for the plugin

**get_option**(*optionname*, *default=0*)
Retrieve the value of the requested option, searching in order: parameters passed from the command line, set via *set_option()*, or the global_plugin_options dict.

*optionname* may be iterable, in which case this function will return the first match.

**Parameters**

- **optionname** (str) – The name of the option to retrieve the value of
- **default** – Optionally provide a default value to return if no option matching *optionname* is found. Default 0

**Returns** The value of *optionname* if found, else *default*

**get_option_as_list**(*optionname*, *delimiter=','*, *default=None*)
Will try to return the option as a list separated by the delimiter.

**get_predicate**(*cmd=False*, *pred=None*)
Get the current default *Plugin* or command predicate.

**Parameters**

- **cmd** (bool) – If a command predicate is set, should it be used.
- **pred** (SoSPredicate) – An optional predicate to pass if no command or plugin predicate is set

**Returns** *pred* if neither a command predicate or plugin predicate is set. The command predicate if one is set and *cmd* is True, else the plugin default predicate (which may be None).

**Return type** SoSPredicate or None

**get_process_pids**(*process*)
Get a list of all PIDs that match a specified name

**Parameters** **process** (str) – The name of the process the get PIDs for

**Returns** A list of PIDs

**Return type** list

**get_service_names**(*regex*)
Get all service names matching regex

**Parameters** **regex** (str) – A regex to match service names against

**Returns** All service name(s) matching the given *regex*

**Return type** list

**get_service_status**(*name*)
Return the reported status for service $name

**Parameters** **name** (str) – The name of the service to check

**Returns** The state of the service according to the init system

**Return type** str

**get_tags_for_cmd**(*cmd*)
Get the tag(s) that should be associated with the given command

**Parameters** **cmd** (str) – The command that tags should be applied to

**Returns** Any tags associated with the command

**Return type** list

**get_tags_for_file**(*fname*)
Get the tags that should be associated with a file matching a given regex

**Parameters** **fname** (str) – A regex for filenames to be matched against

**Returns** The tag(s) associated with *fname*

**Return type** list of strings

**is_installed**(*package_name*)
Is the package $package_name installed?

**Parameters** **package_name** (str) – The name of the package to check

**Returns** True id the package is installed, else False

**Return type** bool

**is_module_loaded**(*module_name*)
Determine whether specified module is loaded or not

**Parameters** **module_name** (str) – Name of kernel module to check for presence

**Returns** True if the module is loaded, else False

**Return type** bool

**is_service**(*name*)
Does the service $name exist on the system?

**Parameters** **name** (str) – The name of the service to check

**Returns** True if service is present on the system, else False

**Return type** bool

**is_service_disabled**(*name*)
Is the service $name disabled?

**Parameters** **name** (str) – The name of the service to check

**Returns** True if service is disabled on the system, else False

**Return type** bool

**is_service_enabled**(*name*)
Is the service $name enabled?

**Parameters** **name** (str) – The name of the service to check

**Returns** True if service is enabled on the system, else ``False

**Return type** bool

**is_service_running**(*name*)
Is the service $name currently running?

**Parameters** **name** (str) – The name of the service to check

**Returns** True if the service is running on the system, else False

**Return type** bool

**join_sysroot**(*path*)
Join a given path with the configured sysroot

**Parameters** **path** (str) – The filesystem path that needs to be joined

**Returns** The joined filesystem path

**Return type** str

**listdir**(*path*)
Helper to call the sos.utilities wrapper that allows the corresponding *os* call to account for sysroot

**Parameters** **path** (str) – The canonical path for a specific file/directory

**Returns** Contents of path, if it is a directory

**Return type** list

**log_skipped_cmd**(*pred*, *cmd*, *kmods=False*, *services=False*, *changes=False*)
Log that a command was skipped due to predicate evaluation.

Emit a warning message indicating that a command was skipped due to predicate evaluation. If kmods or services are True then the list of expected kernel modules or services will be included in the log message. If allow_changes is True a message indicating that the missing data can be collected by using the "–allow-system-changes" command line option will be included.

**Parameters**

- **pred** (SoSPredicate) – The predicate that caused the command to be skipped
- **cmd** (str) – The command that was skipped
- **kmods** (bool) – Did kernel modules cause the command to be skipped
- **services** (bool) – Did services cause the command to be skipped
- **changes** (bool) – Is the *–allow-system-changes* enabled

**classmethod name**()
Get the name of the plugin

**Returns** The name of the plugin, in lowercase

**Return type** str

**path_exists**(*path*)
Helper to call the sos.utilities wrapper that allows the corresponding *os* call to account for sysroot

**Parameters** **path** (str) – The canonical path for a specific file/directory

**Returns** True if the path exists in sysroot, else False

**Return type** bool

**path_isdir**(*path*)
Helper to call the sos.utilities wrapper that allows the corresponding *os* call to account for sysroot

**Parameters** **path** (str) – The canonical path for a specific file/directory

**Returns** True if the path is a dir, else False

**Return type** bool

**path_isfile**(*path*)
Helper to call the sos.utilities wrapper that allows the corresponding *os* call to account for sysroot

**Parameters** **path** (str) – The canonical path for a specific file/directory

**Returns** True if the path is a file, else False

**Return type** bool

**path_islink**(*path*)
Helper to call the sos.utilities wrapper that allows the corresponding *os* call to account for sysroot

**Parameters** **path** (str) – The canonical path for a specific file/directory

**Returns** True if the path is a link, else False

**Return type** bool

**postproc**()
Perform any postprocessing. To be replaced by a plugin if required.

**set_cmd_predicate**(*pred*)
Set or clear the default predicate for command collection for this plugin. If set, this predecate takes precedence over the *Plugin* default predicate for command and journal data collection.

**Parameters** **pred** (SoSPredicate) – The predicate to use as the default command predicate

**set_option**(*optionname*, *value*)
Set the named option to value. Ensure the original type of the option value is preserved

**Parameters**

- **optioname** (str) – The name of the option to set
- **value** – The value to set the option to

**Returns** True if the option is successfully set, else False

**Return type** bool

**set_plugin_manifest**(*manifest*)
Pass in a manifest object to the plugin to write to

**Parameters** **manifest** (SoSManifest) – The manifest that the plugin will add metadata to

**set_predicate**(*pred*)
Set or clear the default predicate for this plugin.

**Parameters** **pred** (SoSPredicate) – The predicate to use as the default for this plugin

**setup**()
Collect the list of files declared by the plugin. This method may be overridden to add further copy_specs, forbidden_paths, and external programs if required.

**strip_sysroot**(*path*)
Remove the configured sysroot from a filesystem path

**Parameters** **path** (str) – The filesystem path to strip sysroot from

**Returns** The stripped filesystem path

**Return type** str

**test_predicate**(*cmd=False*, *pred=None*)
Test the current predicate and return its value.

**Parameters**

- **cmd** – `True` if the predicate is gating a command or `False` otherwise.
- **pred** – An optional predicate to override the current `Plugin` or command predicate.

**Returns** `True` or `False` based on predicate evaluation, or `False` if no predicate

**Return type** `bool`

**timeout**
Returns either the default plugin timeout value, the value as provided on the commandline via -k plugin.timeout=value, or the value of the global –plugin-timeout option.

**timeout_from_options**(*optname*, *plugoptname*, *default_timeout*)
Returns either the default [plugin|cmd] timeout value, the value as provided on the commandline via -k plugin.[|cmd-]timeout=value, or the value of the global –[plugin|cmd]-timeout option.

**tmp_in_sysroot**()
Check if sysroot is within the archive's temp directory

**Returns** `True` if sysroot is in the archive's temp directory, else `False`

**Return type** `bool`

**use_sysroot**()
Determine if the configured sysroot needs to be used

**Returns** `True` if sysroot is not */*, else `False`

**Return type** `bool`

**class** sos.report.plugins.**RedHatPlugin**
Bases: `object`

Tagging class for Red Hat's Linux distributions

**class** sos.report.plugins.**SCLPlugin**
Bases: `sos.report.plugins.RedHatPlugin`

Superclass for plugins operating on Software Collections (SCLs).

Subclasses of this plugin class can specify class.files and class.packages using "%(scl_name)s" interpolation. The plugin invoking mechanism will try to match these against all found SCLs on the system. SCLs that do match class.files or class.packages are then accessible via self.scls_matched when the plugin is invoked.

Additionally, this plugin class provides "add_cmd_output_scl" (run a command in context of given SCL), and "add_copy_spec_scl" and "add_copy_spec_limit_scl" (copy package from file system of given SCL).

For example, you can implement a plugin that will list all global npm packages in every SCL that contains "npm" package:

**class SCLNpmPlugin(Plugin, SCLPlugin):** packages = ("%(scl_name)s-npm",)

**def setup(self):**

**for scl in self.scls_matched:** self.add_cmd_output_scl(scl, "npm ls -g –json")

**add_cmd_output_scl**(*scl*, *cmds*, ***kwargs*)
Same as add_cmd_output, except that it wraps command in "scl enable" call and sets proper PATH.

**add_copy_spec_scl**(*scl*, *copyspecs*)
Same as add_copy_spec, except that it prepends path to SCL root to "copyspecs".

**convert_cmd_scl**(*scl*, *cmd*)
wrapping command in "scl enable" call and adds proper PATH

**class** sos.report.plugins.**SoSCommand**(***kwargs*)
Bases: object

A class to represent a command to be collected.

A SoSCommand() object is instantiated for each command handed to an _add_cmd_output() call, so that we no longer need to pass around a very long tuple to handle the parameters.

Any option supported by _add_cmd_output() is passed to the SoSCommand object and converted to an attribute. SoSCommand.__dict__ is then passed to _get_command_output_now() for each command to be collected.

**class** sos.report.plugins.**SoSPredicate**(*owner*, *dry_run=False*, *kmods=[]*, *services=[]*, *packages=[]*, *cmd_outputs=[]*, *arch=[]*, *required={}*)
Bases: object

A class to implement collection predicates.

A predicate gates the collection of data by an sos plugin. For any *add_cmd_output()*, *add_copy_spec()* or *add_journal()* call, the passed predicate will be evaulated and collection will proceed if the result is *True*, and not otherwise.

Predicates may be used to control conditional data collection without the need for explicit conditional blocks in plugins.

**Parameters**

- **owner** (Plugin) – The Plugin object creating the predicate
- **dry_run** (bool) – Is sos running in dry_run mode?
- **kmods** (list, or str of single name) – Kernel module name(s) to check presence of
- **services** (list, or str of single name) – Service name(s) to check if running
- **packages** (list, or str of single name) – Package name(s) to check presence of
- **cmd_outputs** (list of dict``s, or single ``dict taking form {'cmd': <command to run>, 'output': <string that output should contain>}) – Command to run, with output string to check
- **arch** (list, or str of single architecture) – Architecture(s) that the local system is matched against
- **required** (dict, with keys matching parameter names and values being either 'any', 'all', or 'none. Default 'any'.) – For each parameter provided, should the checks require all items, no items, or any items provided

**arch = []**
Allowed architecture(s) of the system

**dry_run = False**
Skip all collection?

**kmods = []**
Kernel module enablement list

**packages = []**
Package presence list

**report_failure**()
Used by *Plugin()* to obtain the error string based on if the reason was a failed check or a forbidden check

**services = []**
Services enablement list

**class** sos.report.plugins.**SuSEPlugin**
Bases: object

Tagging class for SuSE Linux distributions

**class** sos.report.plugins.**UbuntuPlugin**
Bases: object

Tagging class for Ubuntu Linux

sos.report.plugins.**import_plugin**(*name*, *superclasses=None*)
Import name as a module and return a list of all classes defined in that module. superclasses should be a tuple of valid superclasses to import, this defaults to (Plugin,).

sos.report.plugins.**regex_findall**(*regex*, *fname*)
Return a list of all non overlapping matches in the string(s)

### 6.1.6 sos.report.reporting — Reporting Interface

This provides a restricted tag language to define the sosreport index/report

**class** sos.report.reporting.**Alert**(*content*)
Bases: sos.report.reporting.Leaf

**ADDS_TO = 'alerts'**

**class** sos.report.reporting.**Command**(*name*, *return_code*, *href*)
Bases: sos.report.reporting.Leaf

**ADDS_TO = 'commands'**

**class** sos.report.reporting.**CopiedFile**(*name*, *href*)
Bases: sos.report.reporting.Leaf

**ADDS_TO = 'copied_files'**

**class** sos.report.reporting.**CreatedFile**(*name*, *href*)
Bases: sos.report.reporting.Leaf

**ADDS_TO = 'created_files'**

**class** sos.report.reporting.**HTMLReport**(*report_node*)
Bases: sos.report.reporting.PlainTextReport

Will generate a HTML report from a top_level Report object

**ALERT = '<li>%s</li>'**

**FOOTER = '</body></html>'**

**HEADER = '<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"\n "http://www.w3.or**

**LEAF = '<li><a href="%(href)s">%(name)s</a></li>'**

**NOTE = '<li>%s</li>'**

**PLUGDIVIDER = '<hr/>\n'**

**PLUGINFORMAT = '<h2 id="{name}">Plugin {name}</h2>'**

**PLUGLISTFOOTER = '</tr></table>'**

**PLUGLISTHEADER = '<h3>Loaded Plugins:</h3><table><tr>'**

**PLUGLISTITEM = '<td><a href="#{name}">{name}</a></td>\n'**

**PLUGLISTMAXITEMS = 5**

**PLUGLISTSEP = '</tr>\n<tr>'**

**subsections = ((<class 'sos.report.reporting.Command'>, '<li><a href="%(href)s">%(name**

**class** sos.report.reporting.**JSONReport**(*report_node*)
Bases: sos.report.reporting.PlainTextReport

Will generate a JSON report from a top_level Report object

**unicode**()

**class** sos.report.reporting.**Leaf**
Bases: sos.report.reporting.Node

Marker class that can be added to a Section node

**class** sos.report.reporting.**Node**
Bases: object

**can_add**(*node*)

**class** sos.report.reporting.**Note**(*content*)
Bases: sos.report.reporting.Leaf

**ADDS_TO = 'notes'**

**class** sos.report.reporting.**PlainTextReport**(*report_node*)
Bases: object

Will generate a plain text report from a top_level Report object

**ALERT = ' ! %s'**

**FOOTER = ''**

**HEADER = ''**

**LEAF = ' * %(name)s'**

**NOTE = ' * %s'**

**PLUGDIVIDER = '==========**

**PLUGINFORMAT = '{name}'**

**PLUGLISTFOOTER = ''**

**PLUGLISTHEADER = 'Loaded Plugins:'**

**PLUGLISTITEM = ' {name}'**

**PLUGLISTMAXITEMS = 5**

**PLUGLISTSEP = '\n'**

**line_buf = []**

**process_subsection**(*section*, *key*, *header*, *format_*, *footer*)

**subsections = ((<class 'sos.report.reporting.Command'>, ' * %(name)s', '- commands exe**

**unicode**()

**class** sos.report.reporting.**Report**
Bases: sos.report.reporting.Node

The root element of a report. This is a container for sections.

**add**(**nodes*)

**can_add**(*node*)

**class** sos.report.reporting.**Section**(*name*)
Bases: sos.report.reporting.Node

A section is a container for leaf elements. Sections may be nested inside of Report objects only.

**add**(**nodes*)

**can_add**(*node*)

sos.report.reporting.**ends_bs**(*string*)
Return True if 'string' ends with a backslash, and False otherwise.

Define this as a named function for no other reason than that pep8 now forbids binding of a lambda expression to a name:

'E731 do not assign a lambda expression, use a def'

### 6.1.7 sos.utilities — Utilites Interface

**class** sos.utilities.**AsyncReader**(*channel*, *sizelimit*, *binary*)
Bases: threading.Thread

Used to limit command output to a given size without deadlocking sos.

Takes a sizelimit value in MB, and will compile stdout from Popen into a string that is limited to the given sizelimit.

**get_contents**()
Returns the contents of the deque as a string

**is_full**
Checks if the deque is full, implying that output was truncated

**run**()
Reads from the channel (pipe) that is the output pipe for a called Popen. As we are reading from the pipe, the output is added to a deque. After the size of the deque exceeds the sizelimit earlier (older) entries are removed.

This means the returned output is chunksize-sensitive, but is not really byte-sensitive.

**class** sos.utilities.**ImporterHelper**(*package*)
Bases: object

Provides a list of modules that can be imported in a package. Importable modules are located along the module __path__ list and modules are files that end in .py.

**get_modules**()
Returns the list of importable modules in the configured python package.

**exception** sos.utilities.**SoSTimeoutError**
Bases: OSError

**class** sos.utilities.**TempFileUtil**(*tmp_dir*)
Bases: object

**clean**()

**new**()

sos.utilities.**convert_bytes**(*bytes_*, *K=1024*, *M=1048576*, *G=1073741824*, *T=1099511627776*)
Converts a number of bytes to a shorter, more human friendly format

sos.utilities.**fileobj**(*path_or_file*, *mode='r'*)
Returns a file-like object that can be used as a context manager

sos.utilities.**find**(*file_pattern*, *top_dir*, *max_depth=None*, *path_pattern=None*)
Generator function to find files recursively. Usage:

```
for filename in find("*.properties", "/var/log/foobar"):
    print filename
```

sos.utilities.**get_human_readable**(*size*, *precision=2*)

sos.utilities.**grep**(*pattern*, **files_or_paths*)
Returns lines matched in fnames, where fnames can either be pathnames to files to grep through or open file objects to grep through line by line

sos.utilities.**import_module**(*module_fqname*, *superclasses=None*)
Imports the module module_fqname and returns a list of defined classes from that module. If superclasses is defined then the classes returned will be subclasses of the specified superclass or superclasses. If superclasses is plural it must be a tuple of classes.

sos.utilities.**is_executable**(*command*)
Returns if a command matches an executable on the PATH

sos.utilities.**listdir**(*path*, *sysroot*)

sos.utilities.**path_exists**(*path*, *sysroot*)

sos.utilities.**path_isdir**(*path*, *sysroot*)

sos.utilities.**path_isfile**(*path*, *sysroot*)

sos.utilities.**path_islink**(*path*, *sysroot*)

sos.utilities.**shell_out**(*cmd*, *timeout=30*, *chroot=None*, *runat=None*)
Shell out to an external command and return the output or the empty string in case of error.

sos.utilities.**sos_get_command_output**(*command*, *timeout=300*, *stderr=False*, *chroot=None*, *chdir=None*, *env=None*, *foreground=False*, *binary=False*, *sizelimit=None*, *poller=None*)
Execute a command and return a dictionary of status and output, optionally changing root or current working directory before executing command.

sos.utilities.**tail**(*filename*, *number_of_bytes*)
Returns the last number_of_bytes of filename