

---

## Name

skogpasswdmanapi — library used by skogpasswdmancli and skogpasswdmangui

## Synopsis

```
#!/usr/bin/python

import skogpasswdmanapi

code is_anyst(x)

is_bytestr(x)

is_int(x)

is_num(x)

is_unicodestr(x)

u(x)

b(x)

b2u3(x)

open_rng()

get64(length, pb=None)

get10(length, pb=None)

getint(a, b, pb=None)

unquote(x)

randomize(method, minlength, maxlength, pb=None)

undo(passwdobj=None, honeypotobj=None)

redo(passwdobj=None, honeypotobj=None)

no_pb()

no_pb_f(percent, data)

class progress_bar()

class common_data()

class passwd(common_data)

class honeypot(common_data)
```

**class progress\_bar():**

```
__init__(self, start, stop, function, data=None)
progress(self, percent)
minibar(self, start, stop)
```

**class common\_data():**

```
__init__(self, xmlfile)
__iter__(self)
__next__(self)
next(self)
__getitem__(self, i)
__len__(self)
remove(self, x, xmlfile, element_name, attrib_name, is_numstring=False)
writexml(self, xmlfile, pb=None)
__del__(self)
```

**passwd(common\_data):**

```
__init__(self, backups=True)
add(self, name, value, m_type, m_minlength, m_maxlength, pb=None)
add_nometa(self, name, value)
remove(self, x, is_numstring=False)
__repr__(self)
mkindex(self, x, is_numstring=False)
update(self, index, pb=None)
update_meta(self, index, m_type, m_minlength, m_maxlength, pb=None)
```

**class honeypot(common\_data):**

```
__init__(self, backups=True)
add(self, value)
remove(self, x, is_numstring=False)
pick(self, n=1, sep=",", log_vs_raise=True, pb=None)
pickl(self, n, log_vs_raise=True, pb=None)
pickf(self, n, **arg)
__repr__(self)
```

**Exceptions**

```
class err_norandom(Exception)
class err_nolength(Exception)
class err_loaderr(Exception)
class err_notfound(Exception)
class err_duplicate(Exception)
class err_idiot(Exception)
class err_nometa(Exception)
```

**DESCRIPTION**

Unless otherwise noted, `xmlfile` is a path.

`pb` is either `None` or a `progress_bar` object.

`code` is the encoding `skogpasswdmanapi` will use to encode and decode. It is set on import, but can be changed.

`is_anysr()` returns True if `x` is any kind of string, and False if `x` is not.

`is_bytestr()` returns True if `x` is an encoded string/bytes, and False if `x` is not.

`is_int()` returns True if `x` is an integer, and False if `x` is not.

`is_num()` returns True if `x` is an integer or a float, and False if `x` is not.

`is_unicodestr()` returns True if `x` is a decoded string/unicode, and False if `x` is not.

`u()` returns `x` as a unicode/decoded string.

`b()` returns `x` as a byte/encoded string.

`b2u3()` is the same as `b()` if the Python version is 2.x.

`b2u3()` is the same as `u()` if the Python version is 3.x.

`open_rng()` opens `random(4)` (or `urandom(4)`, if `random` could not be opened). Returns a file open for reading binary. Raises `err_norandom`.

`get10()` and `get64()` returns a random string of `length` letters. `get10()` returns digits. `get64()` returns digits, big letters, small letters, underscores and exclamation marks. Raises `err_norandom` and `err_nolength`.

`getint()` returns a random integer  $\geq a$ ,  $\leq b$ . Raises `err_norandom` and `err_nolength`.

`unquote()` returns the string `x` without its surrounding quotes. If the string is not surrounded by quotes, the string will be returned unchanged.

`randomize()` returns a random string with a `length`  $\geq \text{minlength}$  and  $\leq \text{maxlength}$ . If method is "10" `randomize()` will use `get10()`. If method is "64" `randomize()` will use `get64()`.

`undo()` undoes the latest change to the password list or honey pot list, by restoring from the newest auto-generated backup. It requires `passwdobj` which is the `passwd()` object and `honeypotobj` which is the `honeypot()` object. Raises `err_idiot`.

`redo()` redoes the latest undone change to the password list or honey pot list, by restoring from the newest auto-generated backup from `undo()`. Raises `err_idiot`.

`no_pb()` returns a No-op `progress_bar` object (which calls `no_pb_f()` instead of a function that would actually do something).

`no_pb_f()` does nothing.

`common_data()` is a class defining methods used by both `passwd()` and `honeypot()`.

`passwd()` is a class for the password list. `honeypot()` is a class for the honey-pot list. See FILES.

## **class progress\_bar():**

The class `progress_bar()` is a class for simple interface-independent progress-bars. The keyword-argument `pb` accepts a `progress_bar` object. Only these functions have a `pb` argument:

- `randomize()`
- `getint()`
- `get10()`
- `get64()`
- `common_data.writexml()`

- `passwd.add()`
- `passwd.update()`
- `passwd.update_meta()`
- `honeypot.pick()`
- `honeypot.pickl()`
- `honeypot.pickf()`

`no_pb()` is a useful function.

`progress_bar.__init__()` creates a `progress_bar` object. `start` and `stop` are floats in the range 0...100. `function` is a function that will update the progress-bar. It takes exactly two arguments. The first is the percentage and the other is data which defaults to `None`.

```
#
custom_function(percent, data)
#
```

`percent` is a float in the range 0...100 and `data` is all other necessary data.

`progress()` updates the progress-bar.

`minibar()` creates a new `progress_bar` with identical `function` and `data`. `start` is where in the parent object the child's 0% is and `stop` is where in the parent object the child's 100% is.

## **class common\_data():**

`__init__()` will load the data from `xmlfile`. Raises `err_loaderr`.

`__iter__()` resets the index and returns `self`. `__getitem__()` returns the password/honeypot at `i`. `__len__()` returns the number of passwords/honeypots.

`remove()` removes the password/honeypot at `x`, which can be an integer or a stringed integer or the value of the password/honeypot, from the datastructure `self` and the file `xmlfile`. `element_name` and `attrib_name` tells it what elements in the XML file and attributes it should loop through, remove and find a match for `x` in. Set `is_numstring` to `True` if `x` is a string containing digits. If you don't set it, then `x` will be treated as an index. Raises `err_notfound`.

`writexml()` writes the datastructure `self` to the file `xmlfile`. It creates a backup of `xmlfile` to `~/ .skogpasswdman/undoable`.

## **class passwd(common\_data)**

`passwd()` loads its data from the XML `~/ .skogpasswdman/passwords`.

`self[index]["name"]` is the name/purpose of the password. `self[index]["value"]` is the value of the password. `self[index]["meta"]["minlength"]` is the minimal length required for the password. `self[index]["meta"]["maxlength"]` is the maximal length allowed for the password. `self[index]["meta"]["type"]` is the type of the password, which is one of:

10      The password uses digits.

64      The password uses big letters, small letters, digits, underscores and exclamation marks.

human   The password is human generated.

If a password has no meta-data in `~/.skogpasswdman/passwords`, its `minlength` and `maxlength` will be zero, and its `type` will be "human".

Set `backups` to `False` in `__init__()` if you do not want `passwd()` to make any change undoable (as in can be undone not impossible).

`passwd.add()` and `passwd.add_nometa()` adds a password for `name` with the value `value`. `add_nometa()` adds a password without real meta-data while `add()` requires meta-data (the `m_type` must be a string and `m_minlength` and `m_maxlength` can be either an integer or a stringed integer). `add()` allows `value` to be `None` which will make it randomize a value automatically. Raises `err_duplicate`.

`passwd.remove()` removes the password `x`. `x` can be either a string matching a password's name or an integer (index) or a stringed integer. Set `is_numstring` to `True` if `x` is a string containing digits. If you don't set it, then `x` will be treated as an index. Raises `err_notfound`.

`passwd.mkindex()` find `x` and return an index. `x` can be either a string matching a password's name or a stringed integer (index). Set `is_numstring` to `True` if `x` is a string containing digits. If you don't set it, then `x` will be treated as an index. Raises `err_notfound`.

`passwd.update()` and `passwd.update_meta()` updates the password at `index` automatically by generating a password of the right type and an acceptable length. `update()` uses the password's own meta-data while `update_meta()` gives the password new meta-data from `m_type`, `m_minlength` and `m_maxlength`. `m_type` must be a string, `m_minlength` and `m_maxlength` can be either an integer or a stringed integer. Raises `err_notfound`, `err_idiot` and `err_nometa`.

## class honeypot(common\_data)

The honey pots are weak passwords supposed to only be used as traps. It was a poor choice of name, but it grew legs and glued its feet to the ground. `honeypot()` loads its data from the XML `~/.skogpasswdman/honeypots`. `self[index]` is the value of the honeypot.

Set `backups` to `False` in `__init__()` if you do not want `passwd()` to make any change undoable (as in can be undone not impossible).

`honeypot.add()` adds a new honeypot with the value `value`. Raises `err_duplicate`.

`honeypot.remove()` removes the honeypot `x`. `x` is either an index (integer) or a stringed integer or the value of the honeypot. Set `is_numstring` to `True` if `x` is a string containing digits. If you don't set it, then `x` will be treated as an index. Raises `err_notfound`.

`honeypot.pick()` IS DEPRECATED AND WILL BE REMOVED 2015-01-01. `honeypot.pick()` picks `n` random honeypots and returns a string of honeypots separated with `sep`. If `log_vs_raise` is `true`, it will log an error if `n` is too big. If `log_vs_raise` is `false`, it will raise `err_idiot`.

`honeypot.pickl()` picks `n` random honeypots and returns a list of honeypots. If `log_vs_raise` is `true`, it will log an error if `n` is too big. If `log_vs_raise` is `false`, it will raise `err_idiot`.

`honeypot.pickf()` picks `n` random honeypots and returns a string. Arguments:

<code>n</code>	Required.
<code>pb</code>	Default is <code>None</code> . The progress bar.
<code>pattern</code>	Default is single quote. <code>"(["])"</code>
<code>replacement</code>	Default is to backslash-escape.
<code>sep</code>	Default is single-quote---comma---single-quote. <code>"', '"</code>
<code>head</code>	Default is single quote.

`tail`                      Default is single quote.

`log_vs_raise`            Default is True.

The string is prepended with `head` and appended with `tail`. The honeypots are escaped with the regular expressions `pattern` and `replacement`, and separated with `sep`. If `log_vs_raise` is True then `honeypot.pickf()` will log an error if `n` is too big. It will pick fewer fake-passwords than it is supposed to. If `log_vs_raise` is False it will raise `err_idiot`.

## Exceptions

`err_norandom` is raised when neither `random(4)` or `urandom(4)` can be opened.

- `open_rng()`
- `get10()`
- `get64()`
- `getint()`
- `randomize()`
- `passwd.add()`
- `passwd.update()`
- `passwd.update_meta()`
- `honeypot.pick()`
- `honeypot.pickl()`
- `honeypot.pickf()`

`err_nolength` is raised when a function is called with an invalid length.

- `get64()`
- `get10()`
- `getint()`

`err_loader` is raised if data cannot be loaded from file.

- `common_data()`
- `passwd()`
- `honeypot()`

`err_notfound` is raised if index is out of range or if it cannot find a match.

- `common_data.remove()`
- `passwd.remove()`
- `passwd.mkindex()`
- `passwd.update()`
- `passwd.update_meta()`

- `honeypot.remove()`

`err_duplicate` is raised if it is attempted to add a password with the same name as another or if its is attempted to add a honeypot with the same value as another.

- `passwd.add()`
- `passwd.add_nometa()`
- `honeypot.add()`

`err_idiot` is raised if the function was not used correctly.

- `passwd.update_meta()`
- `honeypot.pick()`
- `honeypot.pickl()`
- `honeypot.pickf()`
- `undo()`
- `redo()`

`err_nometa` is raised when meta-data is required, but the meta-data was nonexistent, corrupt or no good.

- `randomize()`
- `passwd.add()`
- `passwd.update()`

## BUGS

- Single-backup mode actually requires TWO undos.
- `honeypot.pick()` is deprecated and will be removed 2015-01-01. A new `honeypot.pick()` will appear in 2015-02-01.

## FILES

`~/.skogpasswdman/passwords` is the XML file containing the passwords and their meta-data.

`~/.skogpasswdman/honeypots` is the XML file containing the honeypots.

`~/.skogpasswdman/undoable/` is where the auto-generated backups live.

`~/.skogpasswdman/redoable/` is where the backups generated by `undo()` live.

## EXAMPLES

```
$(bindir)/skogpasswdmangui
```

```
$(bindir)/skogpasswdmancli
```

```
$(bindir)/skogpasswdmanrli
```

## AUTHOR

Written by Oskar Skog ([oskar.skog.finland@gmail.com](mailto:oskar.skog.finland@gmail.com)).

Please send patches, questions, bug reports and wish-lists.