# certbot.interfaces module


- certbot.interfaces module


# certbot.interfaces module

Certbot client interfaces.

class certbot.interfaces.AccountStorage[source]
Bases: `object`

Accounts storage interface.

abstractmethod find_all() → list[Account][source]
Find all accounts.

Returns:
All found accounts.

Return type:

list

abstractmethod load(*account_id: str*) → Account[source]
Load an account by its id.

Raises:

- **.AccountNotFound** – if account could not be found
- **.AccountStorageError** – if account could not be loaded

Returns:
The account loaded

Return type:
.Account

abstractmethod save(*account: Account*, *client: ClientV2*) → None[source]
Save account.

Raises:
**.AccountStorageError** – if account could not be saved

class certbot.interfaces.Plugin(*config: NamespaceConfig | None*, *name: str*)[source]
Bases: `object`

Certbot plugin.

Objects providing this interface will be called without satisfying any entry point "extras" (extra dependencies) you might have defined for your plugin, e.g (excerpt from `setup.py` script):

```
setup(
    ...
    entry_points={
        'certbot.plugins': [
            'name=example_project.plugin[plugin_deps]',
        ],
    },
    extras_require={
        'plugin_deps': ['dep1', 'dep2'],
    }
)\
```

Therefore, make sure such objects are importable and usable without extras. This is necessary, because CLI does the following operations (in order):

> - loads an entry point,
> - calls `inject_parser_options`,
> - requires an entry point,
> - creates plugin instance (`__call__`).

description: str = NotImplemented
Short plugin description

name: str = NotImplemented
Unique name of the plugin

abstractmethod prepare() → None[source]
Prepare the plugin.

Finish up any additional initialization.

Raises:

- **.PluginError** – when full initialization cannot be completed.
- **.MisconfigurationError** – when full initialization cannot be completed. Plugin will be displayed on a list of available plugins.
- **.NoInstallationError** – when the necessary programs/files cannot be located. Plugin will NOT be displayed on a list of available plugins.
- **.NotSupportedError** – when the installation is recognized, but the version is not currently supported.

abstractmethod more_info() → str[source]
Human-readable string to help the user.

Should describe the steps taken and any relevant info to help the user decide which plugin to use.

Rtype str:

abstractmethod classmethod inject_parser_options(*parser: ArgumentParser*, *name: str*) → None [source]
Inject argument parser options (flags).

1. Be nice and prepend all options and destinations with `option_namespace` and `dest_namespace`.

2. Inject options (flags) only. Positional arguments are not allowed, as this would break the CLI.

Parameters:

- **parser** (*ArgumentParser*) – (Almost) top-level CLI parser.
- **name** (*str*) – Unique plugin name.

class certbot.interfaces.Authenticator(*config: NamespaceConfig | None*, *name: str*)[source]
Bases: `Plugin`

Generic Certbot Authenticator.

Class represents all possible tools processes that have the ability to perform challenges and attain a certificate.

abstractmethod get_chall_pref(*identifier: str*) → Iterable[type[Challenge]][source]
Return `collections.Iterable` of challenge preferences.

Parameters:

**identifier** (*str*) – Domain or IP address for which challenge preferences are sought.

Returns:
`collections.Iterable` of challenge types (subclasses of `acme.challenges.Challenge`) with the most preferred challenges first. If a type is not specified, it means the Authenticator cannot perform the challenge.

Return type:
`collections.Iterable`

abstractmethod perform(*achalls: list[AnnotatedChallenge]*) → list[ChallengeResponse][source]
Perform the given challenge.

Parameters:

**achalls** (*list*) – Non-empty (guaranteed) list of `AnnotatedChallenge` instances, such that it contains types found within `get_chall_pref()` only.

Returns:
list of ACME `ChallengeResponse` instances corresponding to each provided `Challenge`.

Return type:
`collections.List` of `acme.challenges.ChallengeResponse`, where responses are required to be returned in the same order as corresponding input challenges

Raises:
**.PluginError** – If some or all challenges cannot be performed

abstractmethod cleanup(*achalls: list[AnnotatedChallenge]*) → None[source]
Revert changes and shutdown after challenges complete.

This method should be able to revert all changes made by perform, even if perform exited abnormally.

Parameters:

**achalls** (*list*) – Non-empty (guaranteed) list of `AnnotatedChallenge` instances, a subset of those previously passed to `perform()`.

Raises:

**PluginError** – if original configuration cannot be restored

class certbot.interfaces.Installer(*config: NamespaceConfig | None*, *name: str*)[source]
Bases: `Plugin`

Generic Certbot Installer Interface.

Represents any server that an X509 certificate can be placed.

It is assumed that `save()` is the only method that finalizes a checkpoint. This is important to ensure that checkpoints are restored in a consistent manner if requested by the user or in case of an error.

Using `certbot.reverter.Reverter` to implement checkpoints, rollback, and recovery can dramatically simplify plugin development.

abstractmethod get_all_names() → Iterable[str][source]
Returns all names that may be authenticated.

Return type:
`collections.Iterable` of `str`

abstractmethod deploy_cert(*domain: str*, *cert_path: str*, *key_path: str*, *chain_path: str*, *fullchain_path: str*) → None[source]
Deploy certificate.

Parameters:

- **domain** (*str*) – domain to deploy certificate file
- **cert_path** (*str*) – absolute path to the certificate file
- **key_path** (*str*) – absolute path to the private key file
- **chain_path** (*str*) – absolute path to the certificate chain file
- **fullchain_path** (*str*) – absolute path to the certificate fullchain file (cert plus chain)

Raises:
**.PluginError** – when cert cannot be deployed

abstractmethod enhance(*domain: str*, *enhancement: str*, *options: list[str] | str | None = None*) → None[source]
Perform a configuration enhancement.

Parameters:

- **domain** (*str*) – domain for which to provide enhancement
- **enhancement** (*str*) – An enhancement as defined in `ENHANCEMENTS`

- **options** – Flexible options parameter for enhancement. Check documentation of `ENHANCEMENTS` for expected options for each enhancement.

Raises:
**.PluginError** – If Enhancement is not supported, or if an error occurs during the enhancement.

abstractmethod supported_enhancements() → list[str][source]
Returns a `collections.Iterable` of supported enhancements.

Returns:
supported enhancements which should be a subset of `ENHANCEMENTS`

Return type:
`collections.Iterable` of `str`

abstractmethod save(*title: str | None = None*, *temporary: bool = False*) → None[source]
Saves all changes to the configuration files.

Both title and temporary are needed because a save may be intended to be permanent, but the save is not ready to be a full checkpoint.

It is assumed that at most one checkpoint is finalized by this method. Additionally, if an exception is raised, it is assumed a new checkpoint was not finalized.

Parameters:

- **title** (*str*) – The title of the save. If a title is given, the configuration will be saved as a new checkpoint and put in a timestamped directory. `title` has no effect if temporary is true.
- **temporary** (*bool*) – Indicates whether the changes made will be quickly reversed in the future (challenges)

Raises:
**.PluginError** – when save is unsuccessful

abstractmethod rollback_checkpoints(*rollback: int = 1*) → None[source]
Revert `rollback` number of configuration checkpoints.

Raises:
**.PluginError** – when configuration cannot be fully reverted

abstractmethod recovery_routine() → None[source]
Revert configuration to most recent finalized checkpoint.

Remove all changes (temporary and permanent) that have not been finalized. This is useful to protect against crashes and other execution interruptions.

Raises:
**.errors.PluginError** – If unable to recover the configuration

abstractmethod config_test() → None[source]
Make sure the configuration is valid.

Raises:
**.MisconfigurationError** – when the config is not in a usable state

abstractmethod restart() → None[source]
Restart or refresh the server content.

Raises:
**.PluginError** – when server cannot be restarted

class certbot.interfaces.RenewableCert[source]
Bases: `object`

Interface to a certificate lineage.

abstract property cert_path: str
Path to the certificate file.

Return type:

str

abstract property key_path: str
Path to the private key file.

Return type:

str

abstract property chain_path: str
Path to the certificate chain file.

Return type:

str

abstract property fullchain_path: str
Path to the full chain file.

The full chain is the certificate file plus the chain file.

Return type:

str

abstract property lineagename: str
Name given to the certificate lineage.

Return type:

str

abstractmethod names() → list[str][source]
What are the subject names of this certificate?

Returns:
the subject names

Return type:
`list` of `str`

Raises:
**.CertStorageError** – if could not find cert file.

class certbot.interfaces.GenericUpdater[source]
Bases: `object`

Interface for update types not currently specified by Certbot.

This class allows plugins to perform types of updates that Certbot hasn't defined (yet).

To make use of this interface, the installer should implement the interface methods, and interfaces.GenericUpdater.register(InstallerClass) should be called from the installer code.

The plugins implementing this enhancement are responsible of handling the saving of configuration checkpoints as well as other calls to interface methods of `interfaces.Installer` such as prepare() and restart()

abstractmethod generic_updates(*lineage: RenewableCert*, **args: Any*, ***kwargs: Any*) → None

[source]
Perform any update types defined by the installer.

If an installer is a subclass of the class containing this method, this function will always be called when "certbot renew" is run. If the update defined by the installer should be run conditionally, the installer needs to handle checking the conditions itself.

This method is called once for each lineage.

Parameters:
**lineage** (*RenewableCert*) – Certificate lineage object

class certbot.interfaces.RenewDeployer[source]
Bases: `object`

Interface for update types run when a lineage is renewed

This class allows plugins to perform types of updates that need to run at lineage renewal that Certbot hasn't defined (yet).

To make use of this interface, the installer should implement the interface methods, and interfaces.RenewDeployer.register(InstallerClass) should be called from the installer code.

abstractmethod renew_deploy(*lineage: RenewableCert*, **args: Any*, ***kwargs: Any*) → None[source]
Perform updates defined by installer when a certificate has been renewed

If an installer is a subclass of the class containing this method, this function will always be called when a certificate has been renewed by running "certbot renew". For example if a plugin needs to copy a certificate over, or change configuration based on the new certificate.

This method is called once for each lineage renewed

Parameters:
**lineage** (*RenewableCert*) – Certificate lineage object

class certbot.interfaces.IPluginFactory[source]
Bases: `object`

Compatibility shim for plugins that still use Certbot's old zope.interface classes.

class certbot.interfaces.IPlugin[source]
Bases: `object`

Compatibility shim for plugins that still use Certbot's old zope.interface classes.

class certbot.interfaces.IAuthenticator[source]
Bases: `IPlugin`

Compatibility shim for plugins that still use Certbot's old zope.interface classes.

class certbot.interfaces.IInstaller[source]
Bases: IPlugin

Compatibility shim for plugins that still use Certbot’s old zope.interface classes.

---

□□□ #3
□ □□□ □□□ 2026-10-31 13:02:10 UTC
□ cocobook-sync □□□ 2026-11-05 03:32:34 UTC