

---

# acme-python Documentation

*Release 0*

**Let's Encrypt Project**

**May 02, 2024**



# CONTENTS

<b>1</b>	<b>API Documentation</b>	<b>3</b>
1.1	Challenges . . . . .	3
1.2	Client . . . . .	11
1.3	Errors . . . . .	15
1.4	Fields . . . . .	16
1.5	JOSE . . . . .	17
1.6	Messages . . . . .	17
1.7	Standalone . . . . .	22
<b>2</b>	<b>Indices and tables</b>	<b>25</b>
	<b>Python Module Index</b>	<b>27</b>
	<b>Index</b>	<b>29</b>



Contents:



## API DOCUMENTATION

### 1.1 Challenges

ACME Identifier Validation Challenges.

```
class acme.challenges.Challenge(**kwargs: Any)
    ACME challenge.

    TYPES: Dict[str, Type[Challenge]] = {'dns': <class 'acme.challenges.DNS'>,
    'dns-01': <class 'acme.challenges.DNS01'>, 'http-01': <class
    'acme.challenges.HTTP01'>, 'tls-alpn-01': <class 'acme.challenges.TLSALPN01'>}

    Types registered for JSON deserialization

    classmethod from_json(job: Mapping[str, Any]) → GenericChallenge | UnrecognizedChallenge
        Deserialize ACME object from valid JSON object.

        Raises
        josepy.errors.UnrecognizedTypeError – if type of the ACME object has not been reg-
        istered.

class acme.challenges.ChallengeResponse(**kwargs: Any)
    ACME challenge response.

    TYPES: Dict[str, Type[ChallengeResponse]] = {'dns': <class
    'acme.challenges.DNSResponse'>, 'dns-01': <class 'acme.challenges.DNS01Response'>,
    'http-01': <class 'acme.challenges.HTTP01Response'>, 'tls-alpn-01': <class
    'acme.challenges.TLSALPN01Response'>}

    Types registered for JSON deserialization

    to_partial_json() → Dict[str, Any]
        Get JSON serializable object.

        Returns
        Serializable JSON object representing ACME typed object. validate() will almost cer-
        tainly not work, due to reasons explained in josepy.interfaces.IJSONSerializable.

        Return type
        dict

class acme.challenges.UnrecognizedChallenge(job: Mapping[str, Any])
    Unrecognized challenge.
```

ACME specification defines a generic framework for challenges and defines some standard challenges that are implemented in this module. However, other implementations (including peers) might define additional challenge types, which should be ignored if unrecognized.

**Variables**

**jobj** – Original JSON decoded object.

**to\_partial\_json()** → `Dict[str, Any]`

Get JSON serializable object.

**Returns**

Serializable JSON object representing ACME typed object. `validate()` will almost certainly not work, due to reasons explained in `josepy.interfaces.IJSONSerializable`.

**Return type**

`dict`

**classmethod from\_json(jobj: *Mapping*[*str*, *Any*])** → *UnrecognizedChallenge*

Deserialize ACME object from valid JSON object.

**Raises**

**josepy.errors.UnrecognizedTypeError** – if type of the ACME object has not been registered.

**class acme.challenges.KeyAuthorizationChallengeResponse(\*\*kwargs: *Any*)**

Response to Challenges based on Key Authorization.

**Parameters**

**key\_authorization** (*str*) –

**verify**(*chall*: *KeyAuthorizationChallenge*, *account\_public\_key*: *JWK*) → `bool`

Verify the key authorization.

**Parameters**

- **chall** (*KeyAuthorization*) – Challenge that corresponds to this response.
- **account\_public\_key** (*JWK*) –

**Returns**

True iff verification of the key authorization was successful.

**Return type**

`bool`

**to\_partial\_json()** → `Dict[str, Any]`

Get JSON serializable object.

**Returns**

Serializable JSON object representing ACME typed object. `validate()` will almost certainly not work, due to reasons explained in `josepy.interfaces.IJSONSerializable`.

**Return type**

`dict`

**class acme.challenges.KeyAuthorizationChallenge(\*\*kwargs: *Any*)**

Challenge based on Key Authorization.

**Parameters**

- **response\_cls** – Subclass of *KeyAuthorizationChallengeResponse* that will be used to generate response.
- **typ** (*str*) – type of the challenge



**typ:** `str` = `NotImplemented`

Type of the object. Subclasses must override.

**key\_authorization**(*account\_key*: *JWK*) → *str*

Generate Key Authorization.

**Parameters**

**account\_key** (*JWK*) –

**Rtype** *str*

**response**(*account\_key*: *JWK*) → *KeyAuthorizationChallengeResponse*

Generate response to the challenge.

**Parameters**

**account\_key** (*JWK*) –

**Returns**

Response (initialized *response\_cls*s) to the challenge.

**Return type**

*KeyAuthorizationChallengeResponse*

**abstract validation**(*account\_key*: *JWK*, *\*\*kwargs*: *Any*) → *Any*

Generate validation for the challenge.

Subclasses must implement this method, but they are likely to return completely different data structures, depending on what's necessary to complete the challenge. Interpretation of that return value must be known to the caller.

**Parameters**

**account\_key** (*JWK*) –

**Returns**

Challenge-specific validation.

**response\_and\_validation**(*account\_key*: *JWK*, *\*args*: *Any*, *\*\*kwargs*: *Any*) →

*Tuple*[*KeyAuthorizationChallengeResponse*, *Any*]

Generate response and validation.

Convenience function that return results of *response* and *validation*.

**Parameters**

**account\_key** (*JWK*) –

**Return type**

*tuple*

**class** `acme.challenges.DNS01Response`(*\*\*kwargs*: *Any*)

ACME dns-01 challenge response.

**typ:** `str` = `'dns-01'`

Type of the object. Subclasses must override.

**simple\_verify**(*chall*: *DNS01*, *domain*: *str*, *account\_public\_key*: *JWK*) → *bool*

Simple verify.

This method no longer checks DNS records and is a simple wrapper around *KeyAuthorizationChallengeResponse.verify*.

**Parameters**

- **chall** (*challenges.DNS01*) – Corresponding challenge.

- **domain** (*str*) – Domain name being verified.
- **account\_public\_key** (*JWK*) – Public key for the key pair being authorized.

**Returns**

True iff verification of the key authorization was successful.

**Return type**

*bool*

**class** `acme.challenges.DNS01`(\*\*kwargs: *Any*)

ACME dns-01 challenge.

**response\_cls**

alias of *DNS01Response*

**typ:** *str* = 'dns-01'

Type of the object. Subclasses must override.

**LABEL** = '\_acme-challenge'

Label clients prepend to the domain name being validated.

**validation**(*account\_key*: *JWK*, \*\*unused\_kwargs: *Any*) → *str*

Generate validation.

**Parameters**

**account\_key** (*JWK*) –

**Return type**

*str*

**validation\_domain\_name**(*name*: *str*) → *str*

Domain name for TXT validation record.

**Parameters**

**name** (*str*) – Domain name being validated.

**Return type**

*str*

**class** `acme.challenges.HTTP01Response`(\*\*kwargs: *Any*)

ACME http-01 challenge response.

**typ:** *str* = 'http-01'

Type of the object. Subclasses must override.

**PORT** = 80

Verification port as defined by the protocol.

You can override it (e.g. for testing) by passing `port` to *simple\_verify*.

**WHITESPACE\_CUTSET** = '\n\r\t '

Whitespace characters which should be ignored at the end of the body.

**simple\_verify**(*chall*: *HTTP01*, *domain*: *str*, *account\_public\_key*: *JWK*, *port*: *int* | *None* = *None*, *timeout*: *int* = 30) → *bool*

Simple verify.

**Parameters**

- **chall** (*challenges.SimpleHTTP*) – Corresponding challenge.

- **domain** (*str*) – Domain name being verified.
- **account\_public\_key** (*JWK*) – Public key for the key pair being authorized.
- **port** (*int*) – Port used in the validation.
- **timeout** (*int*) – Timeout in seconds.

**Returns**

True iff validation with the files currently served by the HTTP server is successful.

**Return type**

*bool*

**class** `acme.challenges.HTTP01`(\*\*kwargs: *Any*)

ACME http-01 challenge.

**response\_cls**

alias of *HTTP01Response*

**typ:** *str* = 'http-01'

Type of the object. Subclasses must override.

**URI\_ROOT\_PATH** = '.well-known/acme-challenge'

URI root path for the server provisioned resource.

**property path:** *str*

Path (starting with '/') for provisioned resource.

**Return type**

*str*

**uri**(domain: *str*) → *str*

Create an URI to the provisioned resource.

Forms an URI to the HTTPS server provisioned resource (containing token).

**Parameters**

**domain** (*str*) – Domain name being verified.

**Return type**

*str*

**validation**(account\_key: *JWK*, \*\*unused\_kwargs: *Any*) → *str*

Generate validation.

**Parameters**

**account\_key** (*JWK*) –

**Return type**

*str*

**class** `acme.challenges.TLSALPN01Response`(\*\*kwargs: *Any*)

ACME tls-alpn-01 challenge response.

**typ:** *str* = 'tls-alpn-01'

Type of the object. Subclasses must override.

**PORT** = 443

Verification port as defined by the protocol.

You can override it (e.g. for testing) by passing port to *simple\_verify*.

**property** `h`: `bytes`

Hash value stored in challenge certificate

**gen\_cert**(*domain*: `str`, *key*: `PKey` | `None` = `None`, *bits*: `int` = 2048) → `Tuple`[`X509`, `PKey`]

Generate tls-alpn-01 certificate.

#### Parameters

- **domain** (`str`) – Domain verified by the challenge.
- **key** (`OpenSSL.crypto.PKey`) – Optional private key used in certificate generation. If not provided (`None`), then fresh key will be generated.
- **bits** (`int`) – Number of bits for newly generated key.

#### Return type

`tuple` of `OpenSSL.crypto.X509` and `OpenSSL.crypto.PKey`

**probe\_cert**(*domain*: `str`, *host*: `str` | `None` = `None`, *port*: `int` | `None` = `None`) → `X509`

Probe tls-alpn-01 challenge certificate.

#### Parameters

- **domain** (`str`) – domain being validated, required.
- **host** (`str`) – IP address used to probe the certificate.
- **port** (`int`) – Port used to probe the certificate.

**verify\_cert**(*domain*: `str`, *cert*: `X509`) → `bool`

Verify tls-alpn-01 challenge certificate.

#### Parameters

- **domain** (`str`) – Domain name being validated.
- **cert** (`OpenSSL.crypto.X509`) – Challenge certificate.

#### Returns

Whether the certificate was successfully verified.

#### Return type

`bool`

**simple\_verify**(*chall*: `TLSALPN01`, *domain*: `str`, *account\_public\_key*: `JWK`, *cert*: `X509` | `None` = `None`, *host*: `str` | `None` = `None`, *port*: `int` | `None` = `None`) → `bool`

Simple verify.

Verify validation using `account_public_key`, optionally probe tls-alpn-01 certificate and check using `verify_cert`.

#### Parameters

- **chall** (`.challenges.TLSALPN01`) – Corresponding challenge.
- **domain** (`str`) – Domain name being validated.
- **account\_public\_key** (`JWK`) –
- **cert** (`OpenSSL.crypto.X509`) – Optional certificate. If not provided (`None`) certificate will be retrieved using `probe_cert`.
- **host** (`string`) – IP address used to probe the certificate.
- **port** (`int`) – Port used to probe the certificate.

**Returns**

True if and only if client's control of the domain has been verified.

**Return type**

`bool`

**class** `acme.challenges.TLSALPN01`(*\*\*kwargs: Any*)

ACME tls-alpn-01 challenge.

**response\_cls**

alias of `TLSALPN01Response`

**typ:** `str = 'tls-alpn-01'`

Type of the object. Subclasses must override.

**validation**(*account\_key: JWK, \*\*kwargs: Any*) → `Tuple[X509, PKey]`

Generate validation.

**Parameters**

- **account\_key** (*JWK*) –
- **domain** (*str*) – Domain verified by the challenge.
- **cert\_key** (*OpenSSL.crypto.PKey*) – Optional private key used in certificate generation. If not provided (*None*), then fresh key will be generated.

**Return type**

`tuple` of `OpenSSL.crypto.X509` and `OpenSSL.crypto.PKey`

**static is\_supported**() → `bool`

Check if TLS-ALPN-01 challenge is supported on this machine. This implies that a recent version of OpenSSL is installed ( $\geq 1.0.2$ ), or a recent cryptography version shipped with the OpenSSL library is installed.

**Returns**

True if TLS-ALPN-01 is supported on this machine, False otherwise.

**Return type**

`bool`

**class** `acme.challenges.DNS`(*\*\*kwargs: Any*)

ACME “dns” challenge.

**typ:** `str = 'dns'`

Type of the object. Subclasses must override.

**LABEL** = `'_acme-challenge'`

Label clients prepend to the domain name being validated.

**gen\_validation**(*account\_key: JWK, alg: JWASignature = RS256, \*\*kwargs: Any*) → `JWS`

Generate validation.

**Parameters**

- **account\_key** (*. JWK*) – Private account key.
- **alg** (*. JWA*) –

**Returns**

This challenge wrapped in JWS

**Return type**`.JWS`**check\_validation**(*validation*: *JWS*, *account\_public\_key*: *JWK*) → *bool*

Check validation.

**Parameters**

- **validation** (*JWS*) –
- **account\_public\_key** (*JWK*) –

**Return type**`bool`**gen\_response**(*account\_key*: *JWK*, *\*\*kwargs*: *Any*) → *DNSResponse*

Generate response.

**Parameters**

- **account\_key** (*.JWK*) – Private account key.
- **alg** (*.JWA*) –

**Return type***DNSResponse***validation\_domain\_name**(*name*: *str*) → *str*

Domain name for TXT validation record.

**Parameters****name** (*str*) – Domain name being validated.**class** `acme.challenges.DNSResponse`(*\*\*kwargs*: *Any*)

ACME “dns” challenge response.

**Parameters****validation** (*JWS*) –**typ**: `str = 'dns'`

Type of the object. Subclasses must override.

**check\_validation**(*chall*: *DNS*, *account\_public\_key*: *JWK*) → *bool*

Check validation.

**Parameters**

- **chall** (*challenges.DNS*) –
- **account\_public\_key** (*JWK*) –

**Return type**`bool`

## 1.2 Client

ACME client API.

**class** `acme.client.ClientV2(directory: Directory, net: ClientNetwork)`

ACME client for a v2 API.

### Variables

- **directory** (`messages.Directory`) –
- **net** (`.ClientNetwork`) – Client network.

**new\_account**(*new\_account*: `NewRegistration`) → `RegistrationResource`

Register.

### Parameters

**new\_account** (`.NewRegistration`) –

### Raises

**.ConflictError** – in case the account already exists

### Returns

Registration Resource.

### Return type

`RegistrationResource`

**query\_registration**(*regr*: `RegistrationResource`) → `RegistrationResource`

Query server about registration.

### Parameters

**regr** (`messages.RegistrationResource`) – Existing Registration Resource.

**update\_registration**(*regr*: `RegistrationResource`, *update*: `Registration` | `None` = `None`) → `RegistrationResource`

Update registration.

### Parameters

- **regr** (`messages.RegistrationResource`) – Registration Resource.
- **update** (`messages.Registration`) – Updated body of the resource. If not provided, body will be taken from `regr`.

### Returns

Updated Registration Resource.

### Return type

`RegistrationResource`

**new\_order**(*csr\_pem*: `bytes`) → `OrderResource`

Request a new Order object from the server.

### Parameters

**csr\_pem** (`bytes`) – A CSR in PEM format.

### Returns

The newly created order.

### Return type

`OrderResource`

**poll**(*authzr*: *AuthorizationResource*) → *Tuple[AuthorizationResource, Response]*

Poll Authorization Resource for status.

**Parameters**

**authzr** (*AuthorizationResource*) – Authorization Resource

**Returns**

Updated Authorization Resource and HTTP response.

**Return type**

(*AuthorizationResource*, *requests.Response*)

**poll\_and\_finalize**(*orderr*: *OrderResource*, *deadline*: *datetime* | *None* = *None*) → *OrderResource*

Poll authorizations and finalize the order.

If no deadline is provided, this method will timeout after 90 seconds.

**Parameters**

- **orderr** (*messages.OrderResource*) – order to finalize
- **deadline** (*datetime.datetime*) – when to stop polling and timeout

**Returns**

finalized order

**Return type**

*messages.OrderResource*

**poll\_authorizations**(*orderr*: *OrderResource*, *deadline*: *datetime*) → *OrderResource*

Poll Order Resource for status.

**begin\_finalization**(*orderr*: *OrderResource*) → *OrderResource*

Start the process of finalizing an order.

**Parameters**

- **orderr** (*messages.OrderResource*) – order to finalize
- **deadline** (*datetime.datetime*) – when to stop polling and timeout

**Returns**

updated order

**Return type**

*messages.OrderResource*

**poll\_finalization**(*orderr*: *OrderResource*, *deadline*: *datetime*, *fetch\_alternative\_chains*: *bool* = *False*) → *OrderResource*

Poll an order that has been finalized for its status. If it becomes valid, obtain the certificate.

**Returns**

finalized order (with certificate)

**Return type**

*messages.OrderResource*

**finalize\_order**(*orderr*: *OrderResource*, *deadline*: *datetime*, *fetch\_alternative\_chains*: *bool* = *False*) → *OrderResource*

Finalize an order and obtain a certificate.

**Parameters**

- **orderr** (*messages.OrderResource*) – order to finalize



- **deadline** (*datetime.datetime*) – when to stop polling and timeout
- **fetch\_alternative\_chains** (*bool*) – whether to also fetch alternative certificate chains

**Returns**

finalized order

**Return type**

*messages.OrderResource*

**revoke**(*cert: ComparableX509, rsn: int*) → *None*

Revoke certificate.

**Parameters**

- **cert** (*.ComparableX509*) – OpenSSL.crypto.X509 wrapped in ComparableX509
- **rsn** (*int*) – Reason code for certificate revocation.

**Raises**

**.ClientError** – If revocation is unsuccessful.

**external\_account\_required**() → *bool*

Checks if ACME server requires External Account Binding authentication.

**classmethod get\_directory**(*url: str, net: ClientNetwork*) → *Directory*

Retrieves the ACME directory (RFC 8555 section 7.1.1) from the ACME server. :param str url: the URL where the ACME directory is available :param ClientNetwork net: the ClientNetwork to use to make the request

**Returns**

the ACME directory object

**Return type**

*messages.Directory*

**deactivate\_registration**(*regr: RegistrationResource*) → *RegistrationResource*

Deactivate registration.

**Parameters**

**regr** (*messages.RegistrationResource*) – The Registration Resource to be deactivated.

**Returns**

The Registration resource that was deactivated.

**Return type**

*RegistrationResource*

**deactivate\_authorization**(*authzr: AuthorizationResource*) → *AuthorizationResource*

Deactivate authorization.

**Parameters**

**authzr** (*messages.AuthorizationResource*) – The Authorization resource to be deactivated.

**Returns**

The Authorization resource that was deactivated.

**Return type**

*AuthorizationResource*

**answer\_challenge**(*challb*: *ChallengeBody*, *response*: *ChallengeResponse*) → *ChallengeResource*

Answer challenge.

**Parameters**

- **challb** (*ChallengeBody*) – Challenge Resource body.
- **response** (*challenges.ChallengeResponse*) – Corresponding Challenge response

**Returns**

Challenge Resource with updated body.

**Return type**

*ChallengeResource*

**Raises**

.UnexpectedUpdate –

**classmethod retry\_after**(*response*: *Response*, *default*: *int*) → *datetime*

Compute next *poll* time based on response Retry-After header.

Handles integers and various datestring formats per <https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.37>

**Parameters**

- **response** (*requests.Response*) – Response from *poll*.
- **default** (*int*) – Default value (in seconds), used when Retry-After header is not present or invalid.

**Returns**

Time point when next *poll* should be performed.

**Return type**

*datetime.datetime*

**class acme.client.ClientNetwork**(*key*: *JWK*, *account*: *RegistrationResource* | *None* = *None*, *alg*: *JWASignature* = *RS256*, *verify\_ssl*: *bool* = *True*, *user\_agent*: *str* = *'acme-python'*, *timeout*: *int* = *45*)

Wrapper around requests that signs POSTs for authentication.

Also adds user agent, and handles Content-Type.

**REPLAY\_NONCE\_HEADER** = *'Replay-Nonce'*

Initialize.

**Parameters**

- **key** (*josepy.JWK*) – Account private key
- **account** (*messages.RegistrationResource*) – Account object. Required if you are planning to use *.post()* for anything other than creating a new account; may be set later after registering.
- **alg** (*josepy.JWASignature*) – Algorithm to use in signing JWS.
- **verify\_ssl** (*bool*) – Whether to verify certificates on SSL connections.
- **user\_agent** (*str*) – String to send as User-Agent header.
- **timeout** (*int*) – Timeout for requests.

**head**(\*args: *Any*, \*\*kwargs: *Any*) → Response

Send HEAD request without checking the response.

Note, that `_check_response` is not called, as it is expected that status code other than successfully 2xx will be returned, or `messages2.Error` will be raised by the server.

**get**(url: *str*, content\_type: *str* = 'application/json', \*\*kwargs: *Any*) → Response

Send GET request and check response.

**post**(\*args: *Any*, \*\*kwargs: *Any*) → Response

POST object wrapped in JWS and check response.

If the server responded with a badNonce error, the request will be retried once.

## 1.3 Errors

ACME errors.

**exception** `acme.errors.Error`

Generic ACME error.

**exception** `acme.errors.DependencyError`

Dependency error

**exception** `acme.errors.SchemaValidationError`

JSON schema ACME object validation error.

**exception** `acme.errors.ClientError`

Network error.

**exception** `acme.errors.UnexpectedUpdate`

Unexpected update error.

**exception** `acme.errors.NonceError`

Server response nonce error.

**exception** `acme.errors.BadNonce`(nonce: *str*, error: *Exception*, \*args: *Any*)

Bad nonce error.

**exception** `acme.errors.MissingNonce`(response: *Response*, \*args: *Any*)

Missing nonce error.

According to the specification an “ACME server MUST include an Replay-Nonce header field in each successful response to a POST it provides to a client (...)”.

### Variables

~**.response** (*requests.Response*) – HTTP Response

**exception** `acme.errors.PollError`(exhausted: *Set*[*messages.AuthorizationResource*], updated: *Mapping*[*messages.AuthorizationResource*, *messages.AuthorizationResource*])

Generic error when polling for authorization fails.

This might be caused by either timeout (`exhausted` will be non-empty) or by some authorization being invalid.

### Variables

- **exhausted** – Set of *AuthorizationResource* that didn’t finish within max allowed attempts.

- **updated** – Mapping from original [AuthorizationResource](#) to the most recently updated one

**property timeout:** `bool`

Was the error caused by timeout?

**exception** `acme.errors.ValidationError`(*failed\_authzrs*: [List\[messages.AuthorizationResource\]](#))

Error for authorization failures. Contains a list of authorization resources, each of which is invalid and should have an error field.

**exception** `acme.errors.TimeoutError`

Error for when polling an authorization or an order times out.

**exception** `acme.errors.IssuanceError`(*error*: [messages.Error](#))

Error sent by the server after requesting issuance of a certificate.

**exception** `acme.errors.ConflictError`(*location*: [str](#))

Error for when the server returns a 409 (Conflict) HTTP status.

In the version of ACME implemented by Boulder, this is used to find an account if you only have the private key, but don't know the account URL.

Also used in V2 of the ACME client for the same purpose.

**exception** `acme.errors.WildcardUnsupportedError`

Error for when a wildcard is requested but is unsupported by ACME CA.

## 1.4 Fields

ACME JSON fields.

**class** `acme.fields.Fixed`(*json\_name*: [str](#), *value*: [Any](#))

Fixed field.

**decode**(*value*: [Any](#)) → [Any](#)

Decode a value, optionally with context JSON object.

**encode**(*value*: [Any](#)) → [Any](#)

Encode a value, optionally with context JSON object.

**class** `acme.fields.RFC3339Field`(*json\_name*: [str](#), *default*: [Any](#) = [None](#), *omitempty*: [bool](#) = [False](#), *decoder*: [Callable](#)[[[Any](#)], [Any](#)] | [None](#) = [None](#), *encoder*: [Callable](#)[[[Any](#)], [Any](#)] | [None](#) = [None](#))

RFC3339 field encoder/decoder.

Handles decoding/encoding between RFC3339 strings and aware (not naive) `datetime.datetime` objects (e.g. `datetime.datetime.now(pytz.UTC)`).

**classmethod** `default_encoder`(*value*: [datetime](#)) → [str](#)

Default (passthrough) encoder.

**classmethod** `default_decoder`(*value*: [str](#)) → [datetime](#)

Default decoder.

Recursively deserialize into immutable types ( `josepy.util.frozendict` instead of `dict()`, `tuple()` instead of `list()`).

`acme.fields.fixed(json_name: str, value: Any) → Any`

Generates a type-friendly Fixed field.

`acme.fields.rfc3339(json_name: str, omitempty: bool = False) → Any`

Generates a type-friendly RFC3339 field.

## 1.5 JOSE

The `acme.jose` module was moved to its own package “`josepy`”. Please refer to its documentation there.

## 1.6 Messages

ACME protocol messages.

`acme.messages.is_acme_error(err: BaseException) → bool`

Check if argument is an ACME error.

**class** `acme.messages.IdentifierType(name: str)`

ACME identifier type.

**class** `acme.messages.Identifier(**kwargs: Any)`

ACME identifier.

### Variables

- **typ** (`IdentifierType`) –
- **value** (`str`) –

**exception** `acme.messages.Error(**kwargs: Any)`

ACME error.

<https://datatracker.ietf.org/doc/html/rfc7807>

Note: Although `Error` inherits from `JSONObjectWithFields`, which is immutable, we add mutability for `Error` to comply with the Python exception API.

### Variables

- **typ** (`str`) –
- **title** (`str`) –
- **detail** (`str`) –
- **identifier** (`Identifier`) –
- **subproblems** (`tuple`) – An array of ACME Errors which may be present when the CA returns multiple errors related to the same request, `tuple` of `Error`.

**classmethod** `with_code(code: str, **kwargs: Any) → Error`

Create an `Error` instance with an ACME Error code.

### Str code

An ACME error code, like ‘`dnssec`’.

### Kwargs

kwargs to pass to `Error`.

**property description:** `str` | `None`

Hardcoded error description based on its type.

**Returns**

Description if standard ACME error or `None`.

**Return type**

`str`

**property code:** `str` | `None`

ACME error code.

Basically `self.typ` without the `ERROR_PREFIX`.

**Returns**

error code if standard ACME code or `None`.

**Return type**

`str`

**class** `acme.messages.Status(name: str)`

ACME “status” field.

**class** `acme.messages.Directory(jobj: Mapping[str, Any])`

Directory.

Directory resources must be accessed by the exact field name in RFC8555 (section 9.7.5).

**class** `Meta(**kwargs: Any)`

Directory Meta.

**property terms\_of\_service:** `str`

URL for the CA TOS

**to\_partial\_json()** → `Dict[str, Any]`

Partially serialize.

Following the example, **partial serialization** means the following:

```
assert isinstance(Bar().to_partial_json()[0], Foo)
assert isinstance(Bar().to_partial_json()[1], Foo)

# in particular...
assert Bar().to_partial_json() != ['foo', 'foo']
```

**Raises**

`josepy.errors.SerializationError` – in case of any serialization error.

**Returns**

Partially serializable object.

**classmethod from\_json(jobj: MutableMapping[str, Any])** → `Directory`

Deserialize a decoded JSON document.

**Parameters**

**jobj** – Python object, composed of only other basic data types, as decoded from JSON document. Not necessarily `dict` (as decoded from “JSON object” document).

**Raises**

**josepy.errors.DeserializationError** – if decoding was unsuccessful, e.g. in case of unparseable X509 certificate, or wrong padding in JOSE base64 encoded string, etc.

**class** acme.messages.Resource(\*\*kwargs: Any)

ACME Resource.

**Variables**

**body** (acme.messages.ResourceBody) – Resource body.

**class** acme.messages.ResourceWithURI(\*\*kwargs: Any)

ACME Resource with URI.

**Variables**

**uri** (str) – Location of the resource.

**class** acme.messages.ResourceBody(\*\*kwargs: Any)

ACME Resource Body.

**class** acme.messages.ExternalAccountBinding

ACME External Account Binding

**classmethod** from\_data(account\_public\_key: JWK, kid: str, hmac\_key: str, directory: Directory) → Dict[str, Any]

Create External Account Binding Resource from contact details, kid and hmac.

**class** acme.messages.Registration(\*\*kwargs: Any)

Registration Resource Body.

**Variables**

- **key** (jose.JWK) – Public key.
- **contact** (tuple) – Contact information following ACME spec, tuple of str.
- **agreement** (str) –

**classmethod** from\_data(phone: str | None = None, email: str | None = None, external\_account\_binding: Dict[str, Any] | None = None, \*\*kwargs: Any) → GenericRegistration

Create registration resource from contact details.

The contact keyword being passed to a Registration object is meaningful, so this function represents empty iterables in its kwargs by passing on an empty tuple.

**to\_partial\_json**() → Dict[str, Any]

Modify josepy.JSONDeserializable.to\_partial\_json()

**fields\_to\_partial\_json**() → Dict[str, Any]

Modify josepy.JSONObjectWithFields.fields\_to\_partial\_json()

**property** phones: Tuple[str, ...]

All phones found in the contact field.

**property** emails: Tuple[str, ...]

All emails found in the contact field.

**class** acme.messages.NewRegistration(\*\*kwargs: Any)

New registration.

```
class acme.messages.UpdateRegistration(**kwargs: Any)
```

Update registration.

```
class acme.messages.RegistrationResource(**kwargs: Any)
```

Registration Resource.

#### Variables

- **body** (`acme.messages.Registration`) –
- **new\_authzr\_uri** (`str`) – Deprecated. Do not use.
- **terms\_of\_service** (`str`) – URL for the CA TOS.

```
class acme.messages.ChallengeBody(**kwargs: Any)
```

Challenge Resource Body.

#### Variables

- **acme.challenges.Challenge** – Wrapped challenge. Conveniently, all challenge fields are proxied, i.e. you can call `challb.x` to get `challb.chall.x` contents.
- **status** (`acme.messages.Status`) –
- **validated** (`datetime.datetime`) –
- **error** (`messages.Error`) –

```
encode(name: str) → Any
```

Encode a single field.

#### Parameters

**name** (`str`) – Name of the field to be encoded.

#### Raises

- **errors.SerializationError** – if field cannot be serialized
- **errors.Error** – if field could not be found

```
to_partial_json() → Dict[str, Any]
```

Partially serialize.

Following the example, **partial serialization** means the following:

```
assert isinstance(Bar().to_partial_json()[0], Foo)
assert isinstance(Bar().to_partial_json()[1], Foo)

# in particular...
assert Bar().to_partial_json() != ['foo', 'foo']
```

#### Raises

**josepy.errors.SerializationError** – in case of any serialization error.

#### Returns

Partially serializable object.

```
classmethod fields_from_json(job: Mapping[str, Any]) → Dict[str, Any]
```

Deserialize fields from JSON.



**property uri:** `str`

The URL of this challenge.

**class** `acme.messages.ChallengeResource(**kwargs: Any)`

Challenge Resource.

#### Variables

- **body** (`acme.messages.ChallengeBody`) –
- **authzr\_uri** (`str`) – URI found in the ‘up’ Link header.

**property uri:** `str`

The URL of the challenge body.

**class** `acme.messages.Authorization(**kwargs: Any)`

Authorization Resource Body.

#### Variables

- **identifier** (`acme.messages.Identifier`) –
- **challenges** (`list`) – list of `ChallengeBody`
- **status** (`acme.messages.Status`) –
- **expires** (`datetime.datetime`) –

**class** `acme.messages.NewAuthorization(**kwargs: Any)`

New authorization.

**class** `acme.messages.UpdateAuthorization(**kwargs: Any)`

Update authorization.

**class** `acme.messages.AuthorizationResource(**kwargs: Any)`

Authorization Resource.

#### Variables

- **body** (`acme.messages.Authorization`) –
- **new\_cert\_uri** (`str`) – Deprecated. Do not use.

**class** `acme.messages.CertificateRequest(**kwargs: Any)`

ACME newOrder request.

#### Variables

**csr** (`jose.ComparableX509`) – `OpenSSL.crypto.X509Req` wrapped in `ComparableX509`

**class** `acme.messages.CertificateResource(**kwargs: Any)`

Certificate Resource.

#### Variables

- **body** (`josepy.util.ComparableX509`) – `OpenSSL.crypto.X509` wrapped in `ComparableX509`
- **cert\_chain\_uri** (`str`) – URI found in the ‘up’ Link header
- **authzrs** (`tuple`) – tuple of `AuthorizationResource`.

**class** `acme.messages.Revocation`(\*\*kwargs: *Any*)

Revocation message.

**Variables**

**certificate** (*jose.ComparableX509*) – OpenSSL.crypto.X509 wrapped in jose.ComparableX509

**class** `acme.messages.Order`(\*\*kwargs: *Any*)

Order Resource Body.

**Variables**

- **identifiers** (list of *Identifier*) – List of identifiers for the certificate.
- **status** (*acme.messages.Status*) –
- **authorizations** (list of *str*) – URLs of authorizations.
- **certificate** (*str*) – URL to download certificate as a fullchain PEM.
- **finalize** (*str*) – URL to POST to to request issuance once all authorizations have “valid” status.
- **expires** (*datetime.datetime*) – When the order expires.
- **error** (*Error*) – Any error that occurred during finalization, if applicable.

**class** `acme.messages.OrderResource`(\*\*kwargs: *Any*)

Order Resource.

**Variables**

- **body** (*acme.messages.Order*) –
- **csr\_pem** (*bytes*) – The CSR this Order will be finalized with.
- **authorizations** (list of *acme.messages.AuthorizationResource*) – Fully-fetched AuthorizationResource objects.
- **fullchain\_pem** (*str*) – The fetched contents of the certificate URL produced once the order was finalized, if it’s present.
- **alternative\_fullchains\_pem** (list of *str*) – The fetched contents of alternative certificate chain URLs produced once the order was finalized, if present and requested during finalization.

**class** `acme.messages.NewOrder`(\*\*kwargs: *Any*)

New order.

## 1.7 Standalone

Support for standalone client challenge solvers.

**class** `acme.standalone.TLSServer`(\*args: *Any*, \*\*kwargs: *Any*)

Generic TLS Server.

**server\_bind()** → *None*

Called by constructor to bind the socket.

May be overridden.

**class** `acme.standalone.ACMEServerMixin`

ACME server common settings mixin.

**class** `acme.standalone.BaseDualNetworkedServers`(*ServerClass: Type[TCPServer]*, *server\_address: Tuple[str, int]*, *\*remaining\_args: Any*, *\*\*kwargs: Any*)

Base class for a pair of IPv6 and IPv4 servers that tries to do everything it's asked for both servers, but where failures in one server don't affect the other.

If two servers are instantiated, they will serve on the same port.

**serve\_forever**() → `None`

Wraps `socketserver.TCPServer.serve_forever`

**getsocknames**() → `List[Tuple[str, int]]`

Wraps `socketserver.TCPServer.socket.getsockname`

**shutdown\_and\_server\_close**() → `None`

Wraps `socketserver.TCPServer.shutdown`, `socketserver.TCPServer.server_close`, and `threading.Thread.join`

**class** `acme.standalone.TLSALPN01Server`(*server\_address: Tuple[str, int]*, *certs: List[Tuple[PKey, X509]]*, *challenge\_certs: Mapping[bytes, Tuple[PKey, X509]]*, *ipv6: bool = False*)

TLSALPN01 Server.

**class** `acme.standalone.HTTPServer`(*\*args: Any*, *\*\*kwargs: Any*)

Generic HTTP Server.

**class** `acme.standalone.HTTP01Server`(*server\_address: Tuple[str, int]*, *resources: Set[HTTP01]*, *ipv6: bool = False*, *timeout: int = 30*)

HTTP01 Server.

**class** `acme.standalone.HTTP01DualNetworkedServers`(*\*args: Any*, *\*\*kwargs: Any*)

HTTP01Server Wrapper. Tries everything for both. Failures for one don't affect the other.

**class** `acme.standalone.HTTP01RequestHandler`(*\*args: Any*, *\*\*kwargs: Any*)

HTTP01 challenge handler.

Adheres to the `stdlib's socketserver.BaseRequestHandler` interface.

**Variables**

**simple\_http\_resources** (`set`) – A set of `HTTP01Resource` objects. TODO: better name?

**class** `HTTP01Resource`(*chall, response, validation*)

**chall**

Alias for field number 0

**response**

Alias for field number 1

**validation**

Alias for field number 2

**property timeout:** `int`

The default timeout this server should apply to requests. :return: timeout to apply :rtype: int

**log\_message**(*format: str*, *\*args: Any*) → `None`

Log arbitrary message.

**handle()** → [None](#)

Handle request.

**handle\_index()** → [None](#)

Handle index page.

**handle\_404()** → [None](#)

Handler 404 Not Found errors.

**handle\_simple\_http\_resource()** → [None](#)

Handle HTTP01 provisioned resources.

**classmethod partial\_init**(*simple\_http\_resources: Set[HTTP01], timeout: int*) →  
partial[[HTTP01RequestHandler](#)]

Partially initialize this handler.

This is useful because [socketserver.BaseServer](#) takes uninitialized handler and initializes it with the current request.

ACME protocol implementation.

This module is an implementation of the [ACME](#) protocol.

## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### a

- `acme`, [24](#)
- `acme.challenges`, [3](#)
- `acme.client`, [11](#)
- `acme.errors`, [15](#)
- `acme.fields`, [16](#)
- `acme.messages`, [17](#)
- `acme.standalone`, [22](#)





## A

acme  
 module, 24  
 acme.challenges  
 module, 3  
 acme.client  
 module, 11  
 acme.errors  
 module, 15  
 acme.fields  
 module, 16  
 acme.messages  
 module, 17  
 acme.standalone  
 module, 22  
 ACMEServerMixin (class in acme.standalone), 22  
 answer\_challenge() (acme.client.ClientV2 method), 13  
 Authorization (class in acme.messages), 21  
 AuthorizationResource (class in acme.messages), 21

## B

BadNonce, 15  
 BaseDualNetworkedServers (class in acme.standalone), 23  
 begin\_finalization() (acme.client.ClientV2 method), 12

## C

CertificateRequest (class in acme.messages), 21  
 CertificateResource (class in acme.messages), 21  
 chall (acme.standalone.HTTP01RequestHandler.HTTP01Resource attribute), 23  
 Challenge (class in acme.challenges), 3  
 ChallengeBody (class in acme.messages), 20  
 ChallengeResource (class in acme.messages), 21  
 ChallengeResponse (class in acme.challenges), 3  
 check\_validation() (acme.challenges.DNS method), 10  
 check\_validation() (acme.challenges.DNSResponse method), 10  
 ClientError, 15

ClientNetwork (class in acme.client), 14  
 ClientV2 (class in acme.client), 11  
 code (acme.messages.Error property), 18  
 ConflictError, 16

## D

deactivate\_authorization() (acme.client.ClientV2 method), 13  
 deactivate\_registration() (acme.client.ClientV2 method), 13  
 decode() (acme.fields.Fixed method), 16  
 default\_decoder() (acme.fields.RFC3339Field class method), 16  
 default\_encoder() (acme.fields.RFC3339Field class method), 16  
 DependencyError, 15  
 description (acme.messages.Error property), 17  
 Directory (class in acme.messages), 18  
 Directory.Meta (class in acme.messages), 18  
 DNS (class in acme.challenges), 9  
 DNS01 (class in acme.challenges), 6  
 DNS01Response (class in acme.challenges), 5  
 DNSResponse (class in acme.challenges), 10

## E

emails (acme.messages.Registration property), 19  
 encode() (acme.fields.Fixed method), 16  
 encode() (acme.messages.ChallengeBody method), 20  
 Error, 15, 17  
 external\_account\_required() (acme.client.ClientV2 method), 13  
 ExternalAccountBinding (class in acme.messages), 19

## F

fields\_from\_json() (acme.messages.ChallengeBody class method), 20  
 fields\_to\_partial\_json() (acme.messages.Registration method), 19  
 finalize\_order() (acme.client.ClientV2 method), 12  
 Fixed (class in acme.fields), 16  
 fixed() (in module acme.fields), 16

`from_data()` (*acme.messages.ExternalAccountBinding* class method), 19

`from_data()` (*acme.messages.Registration* class method), 19

`from_json()` (*acme.challenges.Challenge* class method), 3

`from_json()` (*acme.challenges.UnrecognizedChallenge* class method), 4

`from_json()` (*acme.messages.Directory* class method), 18

## G

`gen_cert()` (*acme.challenges.TLSALPN01Response* method), 8

`gen_response()` (*acme.challenges.DNS* method), 10

`gen_validation()` (*acme.challenges.DNS* method), 9

`get()` (*acme.client.ClientNetwork* method), 15

`get_directory()` (*acme.client.ClientV2* class method), 13

`getsocknames()` (*acme.standalone.BaseDualNetworkedServers* method), 23

## H

`h` (*acme.challenges.TLSALPN01Response* property), 7

`handle()` (*acme.standalone.HTTP01RequestHandler* method), 23

`handle_404()` (*acme.standalone.HTTP01RequestHandler* method), 24

`handle_index()` (*acme.standalone.HTTP01RequestHandler* method), 24

`handle_simple_http_resource()` (*acme.standalone.HTTP01RequestHandler* method), 24

`head()` (*acme.client.ClientNetwork* method), 14

`HTTP01` (class in *acme.challenges*), 7

`HTTP01DualNetworkedServers` (class in *acme.standalone*), 23

`HTTP01RequestHandler` (class in *acme.standalone*), 23

`HTTP01RequestHandler.HTTP01Resource` (class in *acme.standalone*), 23

`HTTP01Response` (class in *acme.challenges*), 6

`HTTP01Server` (class in *acme.standalone*), 23

`HTTPServer` (class in *acme.standalone*), 23

## I

`Identifier` (class in *acme.messages*), 17

`IdentifierType` (class in *acme.messages*), 17

`is_acme_error()` (in module *acme.messages*), 17

`is_supported()` (*acme.challenges.TLSALPN01* static method), 9

`IssuanceError`, 16

## K

`key_authorization()` (*acme.challenges.KeyAuthorizationChallenge* method), 5

`KeyAuthorizationChallenge` (class in *acme.challenges*), 4

`KeyAuthorizationChallengeResponse` (class in *acme.challenges*), 4

## L

`LABEL` (*acme.challenges.DNS* attribute), 9

`LABEL` (*acme.challenges.DNS01* attribute), 6

`log_message()` (*acme.standalone.HTTP01RequestHandler* method), 23

## M

`MissingNonce`, 15

`module`

- acme*, 24
- acme.challenges*, 3
- acme.client*, 11
- acme.errors*, 15
- acme.fields*, 16
- acme.messages*, 17
- acme.standalone*, 22

## N

`new_account()` (*acme.client.ClientV2* method), 11

`new_order()` (*acme.client.ClientV2* method), 11

`NewAuthorization` (class in *acme.messages*), 21

`NewOrder` (class in *acme.messages*), 22

`NewRegistration` (class in *acme.messages*), 19

`NonceError`, 15

## O

`Order` (class in *acme.messages*), 22

`OrderResource` (class in *acme.messages*), 22

## P

`partial_init()` (*acme.standalone.HTTP01RequestHandler* class method), 24

`path` (*acme.challenges.HTTP01* property), 7

`phones` (*acme.messages.Registration* property), 19

`poll()` (*acme.client.ClientV2* method), 11

`poll_and_finalize()` (*acme.client.ClientV2* method), 12

`poll_authorizations()` (*acme.client.ClientV2* method), 12

`poll_finalization()` (*acme.client.ClientV2* method), 12

`PollError`, 15

`PORT` (*acme.challenges.HTTP01Response* attribute), 6

`PORT` (*acme.challenges.TLSALPN01Response* attribute), 7

`post()` (*acme.client.ClientNetwork* method), 15

`probe_cert()` (*acme.challenges.TLSALPN01Response* method), 8

## Q

`query_registration()` (*acme.client.ClientV2* method), 11

## R

`Registration` (class in *acme.messages*), 19

`RegistrationResource` (class in *acme.messages*), 20

`REPLAY_NONCE_HEADER` (*acme.client.ClientNetwork* attribute), 14

`Resource` (class in *acme.messages*), 19

`ResourceBody` (class in *acme.messages*), 19

`ResourceWithURI` (class in *acme.messages*), 19

`response` (*acme.standalone.HTTP01RequestHandler.HTTP01Resource* attribute), 23

`response()` (*acme.challenges.KeyAuthorizationChallenge* method), 5

`response_and_validation()` (*acme.challenges.KeyAuthorizationChallenge* method), 5

`response_cls` (*acme.challenges.DNS01* attribute), 6

`response_cls` (*acme.challenges.HTTP01* attribute), 7

`response_cls` (*acme.challenges.TLSALPN01* attribute), 9

`retry_after()` (*acme.client.ClientV2* class method), 14

`Revocation` (class in *acme.messages*), 21

`revoke()` (*acme.client.ClientV2* method), 13

`rfc3339()` (in module *acme.fields*), 17

`RFC3339Field` (class in *acme.fields*), 16

## S

`SchemaValidationError`, 15

`serve_forever()` (*acme.standalone.BaseDualNetworkedServers* method), 23

`server_bind()` (*acme.standalone.TLSServer* method), 22

`shutdown_and_server_close()` (*acme.standalone.BaseDualNetworkedServers* method), 23

`simple_verify()` (*acme.challenges.DNS01Response* method), 5

`simple_verify()` (*acme.challenges.HTTP01Response* method), 6

`simple_verify()` (*acme.challenges.TLSALPN01Response* method), 8

`Status` (class in *acme.messages*), 18

## T

`terms_of_service` (*acme.messages.Directory.Meta* property), 18

`timeout` (*acme.errors.PollError* property), 16

`timeout` (*acme.standalone.HTTP01RequestHandler* property), 23

`TimeoutError`, 16

`TLSALPN01` (class in *acme.challenges*), 9

`TLSALPN01Response` (class in *acme.challenges*), 7

`TLSALPN01Server` (class in *acme.standalone*), 23

`TLSServer` (class in *acme.standalone*), 22

`to_partial_json()` (*acme.challenges.ChallengeResponse* method), 3

`to_partial_json()` (*acme.challenges.KeyAuthorizationChallengeResponse* method), 4

`to_partial_json()` (*acme.challenges.UnrecognizedChallengeResponse* method), 4

`to_partial_json()` (*acme.messages.ChallengeBody* method), 20

`to_partial_json()` (*acme.messages.Directory* method), 18

`to_partial_json()` (*acme.messages.Registration* method), 19

`typ` (*acme.challenges.DNS* attribute), 9

`typ` (*acme.challenges.DNS01* attribute), 6

`typ` (*acme.challenges.DNS01Response* attribute), 5

`typ` (*acme.challenges.DNSResponse* attribute), 10

`typ` (*acme.challenges.HTTP01* attribute), 7

`typ` (*acme.challenges.HTTP01Response* attribute), 6

`typ` (*acme.challenges.KeyAuthorizationChallenge* attribute), 4

`typ` (*acme.challenges.TLSALPN01* attribute), 9

`typ` (*acme.challenges.TLSALPN01Response* attribute), 7

`TYPES` (*acme.challenges.Challenge* attribute), 3

`TYPES` (*acme.challenges.ChallengeResponse* attribute), 3

## U

`UnexpectedUpdate`, 15

`UnrecognizedChallenge` (class in *acme.challenges*), 3

`update_registration()` (*acme.client.ClientV2* method), 11

`UpdateAuthorization` (class in *acme.messages*), 21

`UpdateRegistration` (class in *acme.messages*), 19

`uri` (*acme.messages.ChallengeBody* property), 20

`uri` (*acme.messages.ChallengeResource* property), 21

`uri()` (*acme.challenges.HTTP01* method), 7

`URI_ROOT_PATH` (*acme.challenges.HTTP01* attribute), 7

## V

`validation` (*acme.standalone.HTTP01RequestHandler.HTTP01Resource* attribute), 23

`validation()` (*acme.challenges.DNS01* method), 6

`validation()` (*acme.challenges.HTTP01* method), 7

`validation()` (*acme.challenges.KeyAuthorizationChallenge* method), 5

`validation()` (*acme.challenges.TLSALPN01* method), 9

`validation_domain_name()` (*acme.challenges.DNS*  
    *method*), 10  
`validation_domain_name()` (*acme.challenges.DNS01*  
    *method*), 6  
`ValidationError`, 16  
`verify()` (*acme.challenges.KeyAuthorizationChallengeResponse*  
    *method*), 4  
`verify_cert()` (*acme.challenges.TLSALPN01Response*  
    *method*), 8

## W

`WHITESPACE_CUTSET` (*acme.challenges.HTTP01Response*  
    *attribute*), 6  
`WildcardUnsupportedError`, 16  
`with_code()` (*acme.messages.Error* class *method*), 17