Latch SDK for Python

Release 3.0.0

Telefónica Innovación Digital

Mar 05, 2025

CONTENTS:

CHAPTER

ONE

GETTING STARTED

1.1 Installation

```
$ pip install latch-sdk-telefonica
```

1.1.1 Extra requirements groups

requests

This group provides the requirements to use requests library as http backend. It allows you to use *latch_sdk.syncio.requests.Latch* core class.

```
$ pip install latch-sdk-telefonica[requests]
```

httpx

This group provides the requirements to use httpx library as http backend. It allows you to use *latch_sdk.syncio.httpx.Latch* core class and its asynchronous version the core class *latch_sdk.asyncio.httpx.Latch*.

```
$ pip install latch-sdk-telefonica[httpx]
```

aiohttp

This group provides the requirements to use aiohttp library as http backend. It allows you to use `latch_sdk.syncio.aiohttp.Latch` core class.

```
$ pip install latch-sdk-telefonica[aiohttp]
```

cli

This group provides the requirements to use the *command-line interface application*.

```
$ pip install latch-sdk-telefonica[cli]
```

Tip

It is possible to install several extra requirements groups joining them with a colon:

```
$ pip install latch-sdk-telefonica[cli,aiohttp]
```

1.2 How to use

1.2.1 Synchronous environments

On synchronous environments you must use the synchronous *LatchSDK* class and one of synchronous Latch core classes: *latch_sdk.syncio.pure.Latch*, *latch_sdk.syncio.requests.Latch* or *latch_sdk.syncio.httpx.Latch*.

The Latch core class must be instanced using application identifier and secret, and it must used to instance the *LatchSDK*.

```
from latch_sdk.syncio import LatchSDK
from latch_sdk.syncio.pure import Latch

latch = LatchSDK(Latch(app_id=MY_APP_ID, secret=MY_SECRET))

status = latch.status(MY_ACCOUNT_ID)
```

1.2.2 Asynchronous environments

On asynchronous environments you must use the asynchronous *LatchSDK* class and one of asynchronous Latch core classes: *latch_sdk.asyncio.aiohttp.Latch*, or *latch_sdk.asyncio.httpx.Latch*.

The Latch core class must be instanced using application identifier and secret, and it must used to instance the *LatchSDK*.

```
import asyncio

from latch_sdk.asyncio import LatchSDK
from latch_sdk.asyncio.aiohttp import Latch

async main():
    latch = LatchSDK(Latch(app_id=MY_APP_ID, secret=MY_SECRET))

    status = await latch.status(MY_ACCOUNT_ID)


asyncio.run(main())
```

CHAPTER

TWO

API REFERENCE

2.1 Synchronous SDK

Module: *latch_sdk.syncio*

Synchronous LatchSDK class must be used as interface to Latch services on synchronous developments environments. It defines a easy to use API for Latch operations.

It requires a core Latch instance which is responsible to make requests and parse reponses from Latch services. You must choose *the core* what fits better to your developement: a pure python using standard libraries, a core pawered by requests library or a core powered by httpx library.

class latch_sdk.syncio.**LatchSDK**(*core:* LatchSansIO*[*TResponse*]*)

Bases: *LatchSDKSansIO*[*Response*]

Latch SDK synchronous main class.

Parameters
core – Latch core driver.

account_history(
account_id: str,
** (Keyword-only parameters separator (PEP 3102))*,
from_dt: datetime | None = None,
to_dt: datetime | None = None,
) → *HistoryResponse*

Get history account.

Parameters

- **account_id** – The account identifier.
- **from_dt** – Datetime to start from.
- **to_dt** – Datetime limit.

Returns
History response object.

account_lock(
account_id: str,
***,
instance_id: str | None = None,
operation_id: str | None = None,
) → True

Locks the operation.

Parameters

- **account_id** – The account identifier.
- **instance_id** – The instance identifier.
- **operation_id** – The operation identifier.

Returns

True if no error.

account_pair(
token: str,
***,
web3_account: str | None = None,
web3_signature: str | None = None,
common_name: str | None = None,
phone_number: str | None = None,
) → str

Pairs the token provider with a user account.

It could raise *warning exceptions* (Recoverable errors) containing the account identifier on exception property *account_id*.

Parameters

- **token** – The token for pairing the app, generated by the Latch mobile app.
- **web3_account** – The Ethereum-based account address to pairing the app.
- **web3_signature** – A proof-of-ownership signature with the account address.
- **common_name** – Name send by the service provider attached to this pairing. Typically the identity or name of the user in the SP.
- **phone_number** – Phone number associated with the user.

Returns

Account identifier.

account_pair_with_id(
account_id: str,
***,
web3_account: str | None = None,
web3_signature: str | None = None,
common_name: str | None = None,
phone_number: str | None = None,
) → str

Pairs the origin provider with a user account (mail).

It could raise *warning exceptions* (Recoverable errors) containing the account identifier on exception property *account_id*.

Parameters

- **account_id** – The email for the pairing account - only useful in staging
- **web3_account** – The Ethereum-based account address to pairing the app.
- **web3_signature** – A proof-of-ownership signature with the account address.
- **common_name** – Name send by the service provider attached to this pairing. Typically the identity or name of the user in the SP.

- **phone_number** – Phone number associated with the user.

Returns
Account identifier.

account_set_metadata(
account_id: str,
*,
common_name: str | None = None,
phone_number: str | None = None,
) → True

Set account metadata.

Parameters

- **common_name** – Name send by the service provider attached to this pairing. Typically the identity or name of the user in the SP.
- **phone_number** – Phone number associated with the user.

Returns
True if no error.

account_status(
account_id: str,
*,
instance_id: str | None = None,
operation_id: str | None = None,
silent: bool = False,
nootp: bool = False,
otp_code: str | None = None,
otp_message: str | None = None,
phone_number: str | None = None,
location: str | None = None,
) → *Status*

Return operation status for a given accountId and operation while sending some custom data (Like OTP token or a message).

It could raise *warning exceptions* (Recoverable errors) containing the account status object on *status* exception property.

Parameters

- **account_id** – The account identifier which status is going to be retrieved.
- **instance_id** – The instance identifier.
- **operation_id** – The operation identifier which status is going to be retrieved.
- **silent** – True for not sending lock/unlock push notifications to the mobile devices, false otherwise.
- **nootp** – True for not generating a OTP if needed.
- **otp_code** – Set OTP code manually.
- **otp_message** – Set OTP message.
- **phone_number** – Phone number associated with the user.
- **location** – Allowed location.

Returns
Status object.

account_unlock(
account_id: str,
*,
instance_id: str | None = None,
operation_id: str | None = None,
) → True

Unlocks the operation

Parameters

- **account_id** – The account identifier
- **instance_id** – The instance identifier
- **operation_id** – The operation identifier

Returns
True if no error.

account_unpair(*account_id: str*) → True

Unpairs the origin provider with a user account.

Parameters
account_id – The account identifier.

Returns
True if no error.

application_create(
name: str,
*,
two_factor: ExtraFeature *= ExtraFeature.DISABLED*,
lock_on_request: ExtraFeature *= ExtraFeature.DISABLED*,
contact_phone: str = '',
contact_mail: str = '',
) → *ApplicationCreateResponse*

Create a new application

Parameters

- **name** – Name of new application
- **two_factor** – Specifies if the *Two Factor* protection is enabled for this instance.
- **lock_on_request** – Specifies if the *Lock latches on status request* feature is disabled, opt-in or mandatory for this instance.
- **contact_phone** – Contact phone number.
- **contact_mail** – Contact email.

Returns
Application creation information.

application_list() → Iterable[*Application*]

Returns the list of registered applications.

Returns
Application object list.

application_remove(*application_id: str*) → True

Remove a given application.

Parameters
application_id – Application identifier

Returns
True if no error.

application_update(
application_id: str,
*,
name: str | None = None,
two_factor: ExtraFeature | None = None,
lock_on_request: ExtraFeature | None = None,
contact_phone: str | None = None,
contact_mail: str | None = None,
) → True

Modify a given application

Parameters

- **application_id** – Application identifier
- **name** – Name of new application
- **two_factor** – Specifies if the *Two Factor* protection is enabled for this instance.
- **lock_on_request** – Specifies if the *Lock latches on status request* feature is disabled, opt-in or mandatory for this instance.
- **contact_phone** – Contact phone number.
- **contact_mail** – Contact email.

Returns
True if no error.

authorization_control_status(
control_id: str,
*,
location: str | None = None,
send_notifications: bool = True,
) → bool

Get authorization control status.

Parameters

- **control_id** – The Authorization control identifier.
- **location** – Authorized location.
- **send_notification** – Whether to send notification to other participants or not.

Returns
False if authorization control is locked, and True otherwise.

instance_create(
account_id: str,
name: str,
*,
operation_id: str | None = None,
) → str

Create an instance.

Parameters

- **account_id** – The account identifier.
- **name** – The name of the instance.
- **operation_id** – The operation identifier.

Returns
Instance identifier.

instance_list(
account_id: str,
*,
operation_id: str | None = None,
) → Iterable[*Instance*]

Get a list of instances.

Parameters

- **account_id** – The account identifier.
- **operation_id** – The operation identifier.

Returns
Instance object list.

instance_remove(
account_id: str,
instance_id: str,
operation_id: str | None = None,
) → True

Remove the instance.

Parameters

- **account_id** – The account identifier.
- **instance_id** – The instance identifier.
- **operation_id** – The operation identifier.

Returns
True if no error.

instance_update(
account_id: str,
instance_id: str,
*,
operation_id: str | None = None,
name: str | None = None,
two_factor: ExtraFeature | *None = None*,
lock_on_request: ExtraFeature | *None = None*,
) → True

Update an instance.

Parameters

- **account_id** – The account identifier.

- **instance_id** – The instance identifier.
- **operation_id** – The operation identifier.
- **name** – The name of the instance.
- **two_factor** – Specifies if the *Two Factor* protection is enabled for this instance.
- **lock_on_request** – Specifies if the *Lock latches on status request* feature is disabled, opt-in or mandatory for this instance.

Returns
True if no error.

operation_create(
parent_id: str,
name: str,
*,
two_factor: ExtraFeature = *ExtraFeature.DISABLED*,
lock_on_request: ExtraFeature = *ExtraFeature.DISABLED*,
) → *Operation*

Add a new operation.

Parameters

- **parent_id** – identifies the parent of the operation to be created.
- **name** – The name of the operation.
- **two_factor** – Specifies if the *Two Factor* protection is enabled for this operation.
- **lock_on_request** – Specifies if the *Lock latches on status request* feature is disabled, opt-in or mandatory for this operation.

Returns
Operation object.

operation_list(
*,
parent_id: str | *None* = *None*,
) → Iterable[*Operation*]

Get a list of operations.

Parameters
parent_id – To filter by parent operation.

Returns
Operation object list.

operation_remove(*operation_id: str*) → True

Remove an operation.

Parameters
operation_id – The operation identifier.

Returns
True if no error.

operation_update(
operation_id: str,
*,
name: str | *None* = *None*,
two_factor: ExtraFeature | *None* = *None*,

lock_on_request: ExtraFeature | *None* = *None*,
) → True

Update an operation.

Parameters

- **operation_id** – The operation identifier.
- **name** – The name of the operation.
- **two_factor** – Specifies if the *Two Factor* protection is enabled for this operation.
- **lock_on_request** – Specifies if the *Lock latches on status request* feature is disabled, opt-in or mandatory for this operation.

Returns

True if no error.

subscription_load() → *UserSubscription*

Returns the developer's subscription.

Returns

Developer's subscription.

totp_create(*user_id: str*, *common_name: str*) → *TOTP*

Create a Time-based one-time password.

Parameters

- **user_id** – User identifier (mail)
- **common_name** – Name for the Totp

Returns

TOTP object.

totp_load(*totp_id: str*) → *TOTP*

Get data information about a given totp.

Parameters

totp_id – Totp Identifier

Returns

TOTP object.

totp_remove(*totp_id: str*) → True

Remove a totp

Parameters

totp_id – Totp Identifier

Returns

True if no error.

totp_validate(*totp_id: str*, *code: str*) → True

Validate a code from a totp

Parameters

- **totp_id** – Totp Identifier
- **code** – Code generated

Returns

True if no error.

2.1.1 Latch cores

Base abstract core

Module: *latch_sdk.syncio.base*

```
class latch_sdk.syncio.base.BaseLatch(
    app_id: str,
    secret_key: str,
    *,
    api_version: str = '3.0',
    host: str = 'latch.tu.com',
    port: int = 443,
    is_https: bool = True,
    proxy_host: str | None = None,
    proxy_port: int | None = None,
)
```

Bases: `LatchSansIO`[`Response`]

Parameters

- **app_id** – Application identifier.
- **secret_key** – Application secret key.
- **api_version** – Version of API to use.
- **host** – Host name for Latch service.
- **port** – Port number for Latch service.
- **is_https** – Whether to use TLS layer or not.
- **proxy_host** – Proxy host name.
- **proxy_port** – Proxy port number.

Using standard library

Module: *latch_sdk.syncio.pure*

This module implements the core Latch class to be used with standard python library. It uses `HTTPConnection` and `HTTPSConnection` aand it does not require any other package to be installed.

> **Tip**
>
> If your project are no using any external http package, you should use this implementation in order to avoid to install other packages.

```
class latch_sdk.syncio.pure.Latch(
    app_id: str,
    secret_key: str,
    *,
    api_version: str = '3.0',
    host: str = 'latch.tu.com',
    port: int = 443,
    is_https: bool = True,
    proxy_host: str | None = None,
    proxy_port: int | None = None,
)
```

Bases: *BaseLatch*

Latch core class using synchronous pure standard library.

Parameters

- **app_id** – Application identifier.
- **secret_key** – Application secret key.
- **api_version** – Version of API to use.
- **host** – Host name for Latch service.
- **port** – Port number for Latch service.
- **is_https** – Whether to use TLS layer or not.
- **proxy_host** – Proxy host name.
- **proxy_port** – Proxy port number.

Using requests library

Module: *latch_sdk.syncio.requests*

This module implements the core Latch class to be used with requests library. By default, the required packages to use this module are no installed. You must install the extra requirements *requests* to be able to use it:

```
$ pip install latch-sdk-telefonica[requests]
```

Warning

If extra requirements are no satisfied an error will rise on module import.

```
class latch_sdk.syncio.requests.Latch(
    app_id: str,
    secret_key: str,
    *,
    api_version: str = '3.0',
    host: str = 'latch.tu.com',
    port: int = 443,
    is_https: bool = True,
    proxy_host: str | None = None,
    proxy_port: int | None = None,
)
```

Bases: *BaseLatch*

Latch core class using synchronous requests library.

Parameters

- **app_id** – Application identifier.
- **secret_key** – Application secret key.
- **api_version** – Version of API to use.
- **host** – Host name for Latch service.
- **port** – Port number for Latch service.

- **is_https** – Whether to use TLS layer or not.
- **proxy_host** – Proxy host name.
- **proxy_port** – Proxy port number.

Using httpx library

Module: *latch_sdk.syncio.httpx*

This module implements the core Latch class to be used with httpx library. By default, the required packages to use this module are no installed. You must install the extra requirements *httpx* to be able to use it:

```
$ pip install latch-sdk-telefonica[httpx]
```

Warning

If extra requirements are no satisfied an error will rise on module import.

class latch_sdk.syncio.httpx.**Latch**(
app_id: str,
secret_key: str,
*,
api_version: str = '3.0',
host: str = 'latch.tu.com',
port: int = 443,
is_https: bool = True,
proxy_host: str | None = None,
proxy_port: int | None = None,
)

Bases: `BaseLatch`

Latch core class using synchronous httpx library.

Parameters

- **app_id** – Application identifier.
- **secret_key** – Application secret key.
- **api_version** – Version of API to use.
- **host** – Host name for Latch service.
- **port** – Port number for Latch service.
- **is_https** – Whether to use TLS layer or not.
- **proxy_host** – Proxy host name.
- **proxy_port** – Proxy port number.

2.2 Asynchronous SDK

Module: *latch_sdk.asyncio*

Asynchronous LatchSDK class must be used as interface to Latch services on asynchronous developments environments. It defines a easy to use API for Latch operations.

It requires a core Latch instance which is responsible to make requests and parse reponses from Latch services. You must choose *the core* what fits better to your developement: a core pawered by aiohttp library or a core powered by httpx library.

class latch_sdk.asyncio.**LatchSDK**(*core: LatchSansIO[TResponse]*)

Bases: *LatchSDKSansIO*[Awaitable[*Response*]]

Latch SDK asynchronous main class.

Parameters
core – Latch core driver.

async account_history(
account_id: str,
***,
from_dt: datetime | None = None,
to_dt: datetime | None = None,
) → *HistoryResponse*

Get history account.

Parameters

- **account_id** – The account identifier.
- **from_dt** – Datetime to start from.
- **to_dt** – Datetime limit.

Returns
History response object.

async account_lock(
account_id: str,
***,
instance_id: str | None = None,
operation_id: str | None = None,
) → True

Locks the operation.

Parameters

- **account_id** – The account identifier.
- **instance_id** – The instance identifier.
- **operation_id** – The operation identifier.

Returns
True if no error.

async account_pair(
token: str,
***,
web3_account: str | None = None,
web3_signature: str | None = None,
common_name: str | None = None,
phone_number: str | None = None,
) → str

Pairs the token provider with a user account.

It could raise *warning exceptions* (Recoverable errors) containing the account identifier on exception property *account_id*.

Parameters

- **token** – The token for pairing the app, generated by the Latch mobile app.
- **web3_account** – The Ethereum-based account address to pairing the app.
- **web3_signature** – A proof-of-ownership signature with the account address.
- **common_name** – Name send by the service provider attached to this pairing. Typically the identity or name of the user in the SP.
- **phone_number** – Phone number associated with the user.

Returns
Account identifier.

```
async account_pair_with_id(
    account_id: str,
    *,
    web3_account: str | None = None,
    web3_signature: str | None = None,
    common_name: str | None = None,
    phone_number: str | None = None,
) → str
```

Pairs the origin provider with a user account (mail).

It could raise *warning exceptions* (Recoverable errors) containing the account identifier on exception property *account_id*.

Parameters

- **account_id** – The email for the pairing account - only useful in staging
- **web3_account** – The Ethereum-based account address to pairing the app.
- **web3_signature** – A proof-of-ownership signature with the account address.
- **common_name** – Name send by the service provider attached to this pairing. Typically the identity or name of the user in the SP.
- **phone_number** – Phone number associated with the user.

Returns
Account identifier.

```
async account_set_metadata(
    account_id: str,
    *,
    common_name: str | None = None,
    phone_number: str | None = None,
) → True
```

Set account metadata.

Parameters

- **common_name** – Name send by the service provider attached to this pairing. Typically the identity or name of the user in the SP.
- **phone_number** – Phone number associated with the user.

Returns
True if no error.

async account_status(
account_id: str,
*,
instance_id: str | None = None,
operation_id: str | None = None,
silent: bool = False,
nootp: bool = False,
otp_code: str | None = None,
otp_message: str | None = None,
phone_number: str | None = None,
location: str | None = None,
) → *Status*

Return operation status for a given accountId and operation while sending some custom data (Like OTP token or a message).

It could raise *warning exceptions* (Recoverable errors) containing the account status object on *status* exception property.

Parameters

- **account_id** – The account identifier which status is going to be retrieved.
- **instance_id** – The instance identifier.
- **operation_id** – The operation identifier which status is going to be retrieved.
- **silent** – True for not sending lock/unlock push notifications to the mobile devices, false otherwise.
- **nootp** – True for not generating a OTP if needed.
- **otp_code** – Set OTP code manually.
- **otp_message** – Set OTP message.
- **phone_number** – Phone number associated with the user.
- **location** – Allowed location.

Returns
Status object.

async account_unlock(
account_id: str,
*,
instance_id: str | None = None,
operation_id: str | None = None,
) → True

Unlocks the operation

Parameters

- **account_id** – The account identifier
- **instance_id** – The instance identifier
- **operation_id** – The operation identifier

Returns
True if no error.

async account_unpair(*account_id: str*) → True

Unpairs the origin provider with a user account.

Parameters
account_id – The account identifier.

Returns
True if no error.

async application_create(
name: str,
*,
two_factor: ExtraFeature = *ExtraFeature.DISABLED*,
lock_on_request: ExtraFeature = *ExtraFeature.DISABLED*,
contact_phone: str = '',
contact_mail: str = '',
) → *ApplicationCreateResponse*

Create a new application

Parameters

- **name** – Name of new application
- **two_factor** – Specifies if the *Two Factor* protection is enabled for this instance.
- **lock_on_request** – Specifies if the *Lock latches on status request* feature is disabled, opt-in or mandatory for this instance.
- **contact_phone** – Contact phone number.
- **contact_mail** – Contact email.

Returns
Application creation information.

async application_list() → Iterable[*Application*]

Returns the list of registered applications.

Returns
Application object list.

async application_remove(*application_id: str*) → True

Remove a given application.

Parameters
application_id – Application identifier

Returns
True if no error.

async application_update(
application_id: str,
*,
name: str | *None* = *None*,
two_factor: ExtraFeature | *None* = *None*,
lock_on_request: ExtraFeature | *None* = *None*,
contact_phone: str | *None* = *None*,
contact_mail: str | *None* = *None*,
) → True

Modify a given application

Parameters

- **application_id** – Application identifier
- **name** – Name of new application
- **two_factor** – Specifies if the *Two Factor* protection is enabled for this instance.
- **lock_on_request** – Specifies if the *Lock latches on status request* feature is disabled, opt-in or mandatory for this instance.
- **contact_phone** – Contact phone number.
- **contact_mail** – Contact email.

Returns

True if no error.

async authorization_control_status(
control_id: str,
*,
location: str | None = None,
send_notifications: bool = True,
) → bool

Get authorization control status.

Parameters

- **control_id** – The Authorization control identifier.
- **location** – Authorized location.
- **send_notification** – Whether to send notification to other participants or not.

Returns

False if authorization control is locked, and True otherwise.

async instance_create(
account_id: str,
name: str,
*,
operation_id: str | None = None,
) → str

Create an instance.

Parameters

- **account_id** – The account identifier.
- **name** – The name of the instance.
- **operation_id** – The operation identifier.

Returns

Instance identifier.

async instance_list(
account_id: str,
*,
operation_id: str | None = None,
) → Iterable[*Instance*]

Get a list of instances.

Parameters

- **account_id** – The account identifier.
- **operation_id** – The operation identifier.

Returns
Instance object list.

async instance_remove(
account_id: str,
instance_id: str,
operation_id: str | None = None,
) → True

Remove the instance.

Parameters

- **account_id** – The account identifier.
- **instance_id** – The instance identifier.
- **operation_id** – The operation identifier.

Returns
True if no error.

async instance_update(
account_id: str,
instance_id: str,
*,
operation_id: str | None = None,
name: str | None = None,
two_factor: ExtraFeature | None = None,
lock_on_request: ExtraFeature | None = None,
) → True

Update an instance.

Parameters

- **account_id** – The account identifier.
- **instance_id** – The instance identifier.
- **operation_id** – The operation identifier.
- **name** – The name of the instance.
- **two_factor** – Specifies if the *Two Factor* protection is enabled for this instance.
- **lock_on_request** – Specifies if the *Lock latches on status request* feature is disabled, opt-in or mandatory for this instance.

Returns
True if no error.

async operation_create(
parent_id: str,
name: str,
*,
two_factor: ExtraFeature = ExtraFeature.DISABLED,
lock_on_request: ExtraFeature = ExtraFeature.DISABLED,
) → *Operation*

Add a new operation.

Parameters

- **parent_id** – identifies the parent of the operation to be created.
- **name** – The name of the operation.
- **two_factor** – Specifies if the *Two Factor* protection is enabled for this operation.
- **lock_on_request** – Specifies if the *Lock latches on status request* feature is disabled, opt-in or mandatory for this operation.

Returns
Operation object.

async operation_list(
*,
parent_id: str | None = None,
) → Iterable[*Operation*]

Get a list of operations.

Parameters
parent_id – To filter by parent operation.

Returns
Operation object list.

async operation_remove(*operation_id: str*) → True

Remove an operation.

Parameters
operation_id – The operation identifier.

Returns
True if no error.

async operation_update(
operation_id: str,
*,
name: str | None = None,
two_factor: ExtraFeature | *None = None*,
lock_on_request: ExtraFeature | *None = None*,
) → True

Update an operation.

Parameters

- **operation_id** – The operation identifier.
- **name** – The name of the operation.
- **two_factor** – Specifies if the *Two Factor* protection is enabled for this operation.
- **lock_on_request** – Specifies if the *Lock latches on status request* feature is disabled, opt-in or mandatory for this operation.

Returns
True if no error.

async subscription_load() → *UserSubscription*

Returns the developer's subscription.

Returns
Developer's subscription.

async totp_create(*user_id: str*, *common_name: str*) → *TOTP*

Create a Time-based one-time password.

Parameters

- **user_id** – User identifier (mail)
- **common_name** – Name for the Totp

Returns
TOTP object.

async totp_load(*totp_id: str*) → *TOTP*

Get data information about a given totp.

Parameters
totp_id – Totp Identifier

Returns
TOTP object.

async totp_remove(*totp_id: str*) → True

Remove a totp

Parameters
totp_id – Totp Identifier

Returns
True if no error.

async totp_validate(*totp_id: str*, *code: str*) → True

Validate a code from a totp

Parameters

- **totp_id** – Totp Identifier
- **code** – Code generated

Returns
True if no error.

2.2.1 Latch cores

Base abstract core

Module: *latch_sdk.asyncio.base*

```
class latch_sdk.asyncio.base.BaseLatch(
    app_id: str,
    secret_key: str,
    *,
    api_version: str = '3.0',
    host: str = 'latch.tu.com',
    port: int = 443,
    is_https: bool = True,
    proxy_host: str | None = None,
    proxy_port: int | None = None,
)
```

Bases: LatchSansIO[Awaitable[Response]]

Parameters

- **app_id** – Application identifier.
- **secret_key** – Application secret key.
- **api_version** – Version of API to use.
- **host** – Host name for Latch service.
- **port** – Port number for Latch service.
- **is_https** – Whether to use TLS layer or not.
- **proxy_host** – Proxy host name.
- **proxy_port** – Proxy port number.

Using aiohttp library

Module: *latch_sdk.asyncio.aiohttp*

This module implements the core Latch class to be used with aiohttp library. By default, the required packages to use this module are no installed. You must install the extra requirements *aiohttp* to be able to use it:

```
$ pip install latch-sdk-telefonica[aiohttp]
```

Warning

If extra requirements are no satisfied an error will rise on module import.

class latch_sdk.asyncio.aiohttp.**Latch**(
app_id: str,
secret_key: str,
***,
api_version: str = '3.0',
host: str = 'latch.tu.com',
port: int = 443,
is_https: bool = True,
proxy_host: str | None = None,
proxy_port: int | None = None,
)

Bases: `BaseLatch`

Latch core class using asynchronous aiohttp library.

Parameters

- **app_id** – Application identifier.
- **secret_key** – Application secret key.
- **api_version** – Version of API to use.
- **host** – Host name for Latch service.
- **port** – Port number for Latch service.
- **is_https** – Whether to use TLS layer or not.
- **proxy_host** – Proxy host name.
- **proxy_port** – Proxy port number.

Using httpx library

Module: *latch_sdk.asyncio.httpx*

This module implements the core Latch class to be used with httpx library. By default, the required packages to use this module are no installed. You must install the extra requirements *httpx* to be able to use it:

```
$ pip install latch-sdk-telefonica[httpx]
```

> **Warning**
>
> If extra requirements are no satisfied an error will rise on module import.

class latch_sdk.asyncio.httpx.**Latch**(
app_id: str,
secret_key: str,
***,
api_version: str = '3.0',
host: str = 'latch.tu.com',
port: int = 443,
is_https: bool = True,
proxy_host: str | None = None,
proxy_port: int | None = None,
)

Bases: `BaseLatch`

Latch core class using asynchronous httpx library.

Parameters

- **app_id** – Application identifier.
- **secret_key** – Application secret key.
- **api_version** – Version of API to use.
- **host** – Host name for Latch service.
- **port** – Port number for Latch service.
- **is_https** – Whether to use TLS layer or not.
- **proxy_host** – Proxy host name.
- **proxy_port** – Proxy port number.

2.3 Models

Module: *latch_sdk.models*

Transfer data models.

class latch_sdk.models.**TwoFactor**(*token: str*, *generated: datetime*)

Bases: `object`

token: `str`

Token to validate.

generated: `datetime`

When the token was generated.

classmethod build_from_dict(*data: dict[str, Any]*, ***kwargs*) → Self

Builds model from dict

class latch_sdk.models.**Status**(
operation_id: str,
status: bool,
two_factor: TwoFactor | *None* = *None*,
operations: 'Iterable[Status] | None' = *None*,
)

Bases: `object`

operation_id: `str`

Operation identifier.

status: `bool`

False means the latch is closed and any action must be blocked.

two_factor: *TwoFactor* | `None` = **None**

Two factor data if it is required.

operations: `Iterable`[*Status*] | `None` = **None**

List of descendant operations.

classmethod build_from_dict(*data: dict[str, Any]*, ***kwargs*) → Self

Builds model from dict

class latch_sdk.models.**ExtraFeature**(**values*)

Bases: `str`, `Enum`

MANDATORY = 'MANDATORY'

Feature is required.

OPT_IN = 'OPT_IN'

Feature is optional.

DISABLED = 'DISABLED'

Feature is disabled.

class latch_sdk.models.**ExtraFeatureStatus**(**values*)

Bases: `str`, `Enum`

MANDATORY = 'MANDATORY'

Feature is required.

OPT_IN = 'OPT_IN'

Feature is optional.

DISABLED = 'DISABLED'

Feature is disabled.

ON = 'on'

Current status of feature is ON.

OFF = 'off'

Current status of feature is OFF.

class latch_sdk.models.**Operation**(
operation_id: str,
name: str,
parent_id: str | None = None,
two_factor: ExtraFeatureStatus *= ExtraFeatureStatus.DISABLED*,
lock_on_request: ExtraFeatureStatus *= ExtraFeatureStatus.DISABLED*,
status: bool | None = None,
)

Bases: object

Latch operation

operation_id: str

Operation identifier.

name: str

Operation name

parent_id: str | None = None

Parent identifier

two_factor: *ExtraFeatureStatus* = 'DISABLED'

State of *Two factor* feature.

lock_on_request: *ExtraFeatureStatus* = 'DISABLED'

State of *Lock on request* feature.

status: bool | None = None

False means the latch is close and any action must be blocked.

classmethod build_from_dict(*data: dict[str, Any]*, ***kwargs*) → Self

Builds model from dict

class latch_sdk.models.**Instance**(
instance_id: str,
name: str,
two_factor: ExtraFeatureStatus *= ExtraFeatureStatus.DISABLED*,
lock_on_request: ExtraFeatureStatus *= ExtraFeatureStatus.DISABLED*,
)

Bases: object

Latch instance.

instance_id: str

Instance identifier

name: str

Instance name

two_factor: *ExtraFeatureStatus* = 'DISABLED'

State of *Two factor* feature.

lock_on_request: *ExtraFeatureStatus* = 'DISABLED'

State of *Lock on request* feature.

classmethod build_from_dict(*data: dict[str, Any]*, ***kwargs*) → Self

Builds model from dict

class latch_sdk.models.**HistoryEntry**(
t: datetime,
action: str,
what: str,
was: Any | None,
value: Any | None,
name: str,
ip: str,
)

Bases: object

History entry. It describes something that happened.

t: datetime

When event happened.

action: str

Action name.

what: str

What happened.

was: Any | None

What was the value before action.

value: Any | None

What is the new value.

name: str

Event name

ip: str

IP or user-agent user make action from.

classmethod build_from_dict(*data: dict[str, Any]*, ***kwargs*) → Self

Builds model from dict

class latch_sdk.models.**ApplicationCreateResponse**(*application_id: str*, *secret: str*)

Bases: object

Latch Application create response

application_id: str

Application identifier.

secret: str

Application secret.

classmethod build_from_dict(
data: dict[str, Any],
***kwargs*,
) → Self

Builds model from dict

class latch_sdk.models.**Application**(
application_id: str,
name: str,
description: str,
image_url: str,
secret: str,
autoclose: int,
contact_mail: str | None = None,
contact_phone: str | None = None,
two_factor: ExtraFeature *= ExtraFeature.DISABLED*,
lock_on_request: ExtraFeature *= ExtraFeature.DISABLED*,
*operations: Iterable[*Operation*] | None = None*,
)

Bases: `object`

Latch Application information

application_id: `str`

Application identifier.

name: `str`

Application name.

description: `str`

Application description.

image_url: `str`

URL to application image.

secret: `str`

Application secret.

autoclose: `int`

Minutes to close latch after open. 0 means no autoclose.

contact_mail: `str | None = None`

Contact email.

contact_phone: `str | None = None`

Contact phone number.

two_factor: *`ExtraFeature`* `= 'DISABLED'`

State of *Two factor* feature.

lock_on_request: *`ExtraFeature`* `= 'DISABLED'`

State of *Lock on request* feature.

operations: `Iterable[`*`Operation`*`] | None = None`

List of descendant operations.

classmethod build_from_dict(*data: dict[str, Any]*, ***kwargs*) → Self

Builds model from dict

class latch_sdk.models.**ApplicationHistory**(
application_id: str,
name: str,
description: str,
image_url: str,

status: bool,
paired_on: datetime,
status_last_modified: datetime,
autoclose: int,
contact_mail: str | None = None,
contact_phone: str | None = None,
two_factor: ExtraFeature = ExtraFeature.DISABLED,
lock_on_request: ExtraFeature = ExtraFeature.DISABLED,
operations: Iterable[Operation] | None = None,
)

Bases: `object`

Latch Application information

application_id: `str`
Application identifier.

name: `str`
Application name.

description: `str`
Application description.

image_url: `str`
URL to application image.

status: `bool`
Application current state. False means the latch is closed and any action must be blocked.

paired_on: `datetime`
When it was paired.

status_last_modified: `datetime`
Last time status changed.

autoclose: `int`
Minutes to close latch after open. 0 means no autoclose.

contact_mail: `str | None = None`
Contact email.

contact_phone: `str | None = None`
Contact phone number.

two_factor: *`ExtraFeature`* `= 'DISABLED'`
State of *Two factor* feature.

lock_on_request: *`ExtraFeature`* `= 'DISABLED'`
State of *Lock on request* feature.

operations: `Iterable[`*`Operation`*`] | None = None`
List of descendant operations.

classmethod build_from_dict(
data: dict[str, Any],
***kwargs*,
) → Self
Builds model from dict

class latch_sdk.models.**Client**(*platform: str*, *app: str*)

Bases: object

Client information

platform: str

Client platform: Window, Android, iOS, etc…

app: str

Client application version.

class latch_sdk.models.**HistoryResponse**(
application: ApplicationHistory,
client_version: Iterable[Client],
count: int,
history: Iterable[HistoryEntry],
last_seen: datetime | None = None,
)

Bases: object

Response for history request.

application: *ApplicationHistory*

Application information.

client_version: Iterable[*Client*]

List of client used by user.

count: int

Number of history entries.

history: Iterable[*HistoryEntry*]

List of history entries.

last_seen: datetime | None = None

Last time user has been seen.

classmethod build_from_dict(*data: dict[str, Any]*, ***kwargs*) → Self

Builds model from dict

class latch_sdk.models.**Identity**(*id: str*, *name: str*)

Bases: object

User identity model.

id: str

User identifier.

name: str

User name

classmethod build_from_dict(*data: dict[str, Any]*, ***kwargs*) → Self

Builds model from dict

class latch_sdk.models.**TOTP**(
totp_id: str,
secret: str,
app_id: str,
identity: Identity,

issuer: str,
algorithm: str,
digits: int,
period: int,
created_at: datetime,
disabled_by_subscription_limit: bool,
qr: str,
uri: str,
)

Bases: `object`

totp_id: `str`
TOTP identifier

secret: `str`
Secret used to create one-time tokens.

app_id: `str`
Dependent application.

identity: *Identity*
User identity.

issuer: `str`
Service provider.

algorithm: `str`
Algorithm used to generate tokens.

digits: `int`
Number of digits of each token.

period: `int`
Life time in second for each token.

created_at: `datetime`
When TOTP was created.

disabled_by_subscription_limit: `bool`
Whether TOTP must be disabled when subscription raises limit.

qr: `str`
HTML encode QR image for *uri* field.

uri: `str`
URI representing whole TOTP data.

classmethod build_from_dict(*data: dict[str, Any]*, ***kwargs*) → Self
Builds model from dict

class latch_sdk.models.**ErrorData**

Bases: `TypedDict`

code: `int`
Error code

message: `str`
Error message

class latch_sdk.models.**Error**(*code: int*, *message: str*)

Bases: object

Error model

code: int

Error code

message: str

Error message

classmethod build_from_dict(*data: dict[str, Any]*, ***kwargs*) → Self

Builds model from dict

class latch_sdk.models.**Response**(
data: dict[str, Any] | None = None,
error: Error *| None = None*,
)

Bases: object

This class models a response from any of the endpoints in the Latch API. It consists of a "data" and an "error" elements. Although normally only one of them will be present, they are not mutually exclusive, since errors can be non fatal, and therefore a response could have valid information in the data field and at the same time inform of an error.

data: dict[str, Any] | None = None

Response data

error: *Error* | None = None

Response error data

classmethod build_from_dict(*data: dict[str, Any]*, ***kwargs*) → Self

Builds model from dict

raise_on_error() → None

Raise an exception if response contains an error

class latch_sdk.models.**SubscriptionUsage**(*in_use: int*, *limit: int | None = None*)

Bases: object

Subscription feature usage.

in_use: int

How many items in use.

limit: int | None = None

Items limit for feature. None means infinite.

classmethod build_from_dict(
data: dict[str, Any],
***kwargs*,
) → Self

Builds model from dict

class latch_sdk.models.**UserSubscription**(
subscription_id: str,
applications: SubscriptionUsage,
operations: dict[str, SubscriptionUsage*]*,
users: SubscriptionUsage,
)

Bases: object

User subcription details.

subscription_id: str

Subcription identifier.

applications: *SubscriptionUsage*

Application registered.

operations: dict[str, *SubscriptionUsage*]

Operations registered.

users: *SubscriptionUsage*

Users registered.

classmethod build_from_dict(*data: dict[str, Any]*, ***kwargs*) → Self

Builds model from dict

2.4 Exceptions

Module: *latch_sdk.exceptions*

Latch exceptions.

2.4.1 Pairing warnings

Pairing warnings are errors that occurs during pairing operations, but developer could recover execution from them. It means it can continue execution like it never had happened.

```
import logging

from latch_sdk.syncio import LatchSDK
from latch_sdk.syncio.pure import Latch


latch = LatchSDK(Latch(MY_APP_ID, MY_APP_SECRET))

try:
   account_id = latch.account_pair(MY_TOKEN)
except PairingWarning as ex:
   logging.warning(ex)

   account_id = ex.account_id
```

exception latch_sdk.exceptions.**PairingWarning**(*code: int*, *message: str*)

Bases: *LatchWarning*

account_id: str | None = None

exception latch_sdk.exceptions.**ApplicationAlreadyPaired**(*code: int*, *message: str*)

Bases: *PairingWarning*

CODE = 205

exception latch_sdk.exceptions.**AccountPairedButDisabled**(*code: int*, *message: str*)

Bases: *PairingWarning*

CODE = 701

2.4.2 Status warnings

Status warnings are errors that occurs during status operation, but developer could recover execution from them. It means it can continue execution like it never had happened.

```
import logging

from latch_sdk.syncio import LatchSDK
from latch_sdk.syncio.pure import Latch


latch = LatchSDK(Latch(MY_APP_ID, MY_APP_SECRET))

try:
   status = latch.account_status(MY_ACCOUNT_ID)
except StatusWarning as ex:
   logging.warning(ex)

   status = ex.status
```

exception latch_sdk.exceptions.**StatusWarning**(*code: int*, *message: str*)

Bases: *LatchWarning*

status: *Status* | None = None

exception latch_sdk.exceptions.**AccountDisabledBySubscription**(*code: int*, *message: str*)

Bases: *StatusWarning*

CODE = 702

exception latch_sdk.exceptions.**NoSilentNotificationBySubscriptionLimit**(*code: int*, *message: str*)

Bases: *StatusWarning*

CODE = 704

2.4.3 OpenGateway warnings

OpenGateway warnings are *PairingWarning* and *StatusWarning* on pairing and status operations respectively.

exception latch_sdk.exceptions.**OpenGatewayWarning**(*code: int*, *message: str*)

Bases: *PairingWarning*, *StatusWarning*

exception latch_sdk.exceptions.**AccountPairedPhoneNumberVerificationFailed**(*code: int*, *message: str*)

Bases: *OpenGatewayWarning*

CODE = 1200

exception latch_sdk.exceptions.**AccountPairedPhoneNumberLocationFailed**(*code: int*, *message: str*)

Bases: *OpenGatewayWarning*

CODE = 1201

exception latch_sdk.exceptions.**AccountPairedPhoneNumberUserConsentRequired**(*code: int*, *message: str*)

Bases: *OpenGatewayWarning*

CODE = 1202

exception latch_sdk.exceptions.**AccountPairedPhoneNumberUserConsentRejected**(*code: int*, *message: str*)

Bases: *OpenGatewayWarning*

CODE = 1203

2.4.4 Errors

exception latch_sdk.exceptions.**BaseLatchException**(*code: int*, *message: str*)

Bases: Exception

classmethod **build**(
code: int,
message: str,
) → *BaseLatchException*

exception latch_sdk.exceptions.**LatchWarning**(*code: int*, *message: str*)

Bases: *BaseLatchException*

exception latch_sdk.exceptions.**LatchError**(*code: int*, *message: str*)

Bases: *BaseLatchException*

exception latch_sdk.exceptions.**InvalidCredentials**(*code: int*, *message: str*)

Bases: *LatchError*

CODE = 105

exception latch_sdk.exceptions.**UnauthorizedUser**(*code: int*, *message: str*)

Bases: *LatchError*

CODE = 111

exception latch_sdk.exceptions.**UnauthorizedScope**(*code: int*, *message: str*)

Bases: *LatchError*

CODE = 113

exception latch_sdk.exceptions.**AccountNotPaired**(*code: int*, *message: str*)

Bases: *LatchError*

CODE = 201

exception latch_sdk.exceptions.**InvalidAccountName**(*code: int*, *message: str*)

Bases: *LatchError*

CODE = 202

exception latch_sdk.exceptions.**PairingError**(*code: int*, *message: str*)

Bases: *LatchError*

CODE = 203

exception latch_sdk.exceptions.**UnpairingError**(*code: int*, *message: str*)

Bases: *LatchError*

CODE = 204

exception latch_sdk.exceptions.**TokenNotFound**(*code: int*, *message: str*)

Bases: *LatchError*

CODE = 206

exception latch_sdk.exceptions.**AccountDisabled**(*code: int*, *message: str*)

Bases: *LatchError*

CODE = 207

exception latch_sdk.exceptions.**ApplicationNotFound**(*code: int*, *message: str*)

Bases: *LatchError*

CODE = 301

exception latch_sdk.exceptions.**InstanceNotFound**(*code: int*, *message: str*)

Bases: *LatchError*

CODE = 302

exception latch_sdk.exceptions.**MaxActionsExceed**(*code: int*, *message: str*)

Bases: *LatchError*

CODE = 303

exception latch_sdk.exceptions.**NoTOTPServerConfigured**(*code: int*, *message: str*)

Bases: *LatchError*

CODE = 304

exception latch_sdk.exceptions.**TOTPNotFound**(*code: int*, *message: str*)

Bases: *LatchError*

CODE = 305

exception latch_sdk.exceptions.**InvalidTOTP**(*code: int*, *message: str*)

Bases: *LatchError*

CODE = 306

exception latch_sdk.exceptions.**TOPTGeneratingError**(*code: int*, *message: str*)

Bases: *LatchError*

CODE = 307

2.5 Utils

Module: *latch_sdk.utils*

Global utilities.

latch_sdk.utils.**sign_data**(*secret: bytes*, *data: bytes*) → bytes

Signs data using a secret.

Parameters

- **secret** – Secret to use to sign *data*.
- **data** – Data to sign.

latch_sdk.utils.**wraps_and_replace_return**(
meth: Callable[Concatenate[Any, P], Any],
return_type: T,
***,
factory_docs: str | None = None,
) → Callable[[Callable[Concatenate[TSelf, P], T]], Callable[Concatenate[TSelf, P], T]]

Wraps a method and replace return type.

Parameters

- **meth** – Request constructor method.
- **return_type** – Return type for new method. It should be the return type of factory.
- **factory_docs** – Documentation from factory function to be appended to *meth* docstring.

2.6 Sans-IO API

Module: *latch_sdk.sansio*

Base Sansio classes. They implement all logic with on input/output processes.

class latch_sdk.sansio.**TResponse**

Response type variable. It will be *Response* on SyncIO classes and Awaitable [*Response*] on AsyncIO classes.

alias of TypeVar('TResponse', ~typing.Awaitable[~latch_sdk.models.Response], ~latch_sdk.models.Response)

class latch_sdk.sansio.**TReturnType**

Return type for factories.

alias of TypeVar('TReturnType')

latch_sdk.sansio.**P = ~P**

Parameters specification for methods.

class latch_sdk.sansio.**Paths**

Bases: TypedDict

Dictionary of versioned path to endpoint.

account_status: str

Endpoint to check account/operation/instance status.

account_pair: str

Endpoint to pair a latch.

account_pair_with_id: str

Endpoint to pair a latch using user identifier.

account_unpair: str
Endpoint to unpair a latch.

account_lock: str
Endpoint to lock an account.

account_unlock: str
Endpoint to unlock an account.

account_history: str
Endpoint to retrieve account history.

account_metadata: str
Endpoint to change account metadata.

operation: str
Endpoint to manage operations.

subscription: str
Endpoint to manage subscription.

application: str
Endpoint to manage applications.

instance: str
Endpoint to manage instances.

totp: str
Endpoint to manage TOTPs.

authorization_control_status: str
Endpoint to check Authorization control status.

class latch_sdk.sansio.**LatchSansIO**(
app_id: str,
secret_key: str,
***,
api_version: str = '3.0',
host: str = 'latch.tu.com',
port: int = 443,
is_https: bool = True,
proxy_host: str | None = None,
proxy_port: int | None = None,
)

Bases: ABC, Generic[*TResponse*]

Latch core sans-io.

Parameters

- **app_id** – Application identifier.
- **secret_key** – Application secret key.
- **api_version** – Version of API to use.
- **host** – Host name for Latch service.
- **port** – Port number for Latch service.
- **is_https** – Whether to use TLS layer or not.

- **proxy_host** – Proxy host name.
- **proxy_port** – Proxy port number.

API_PATH_CHECK_STATUS_PATTERN = '/api/${version}/status'

API_PATH_PAIR_PATTERN = '/api/${version}/pair'

API_PATH_PAIR_WITH_ID_PATTERN = '/api/${version}/pairWithId'

API_PATH_UNPAIR_PATTERN = '/api/${version}/unpair'

API_PATH_LOCK_PATTERN = '/api/${version}/lock'

API_PATH_UNLOCK_PATTERN = '/api/${version}/unlock'

API_PATH_HISTORY_PATTERN = '/api/${version}/history'

API_PATH_METADATA_PATTERN = '/api/${version}/aliasMetadata'

API_PATH_OPERATION_PATTERN = '/api/${version}/operation'

API_PATH_SUBSCRIPTION_PATTERN = '/api/${version}/subscription'

API_PATH_APPLICATION_PATTERN = '/api/${version}/application'

API_PATH_INSTANCE_PATTERN = '/api/${version}/instance'

API_PATH_TOTP_PATTERN = '/api/${version}/totps'

API_PATH_AUTHORIZATION_CONTROL_STATUS_PATTERN = '/api/${version}/control-status'

AUTHORIZATION_HEADER_NAME = 'Authorization'

DATE_HEADER_NAME = 'X-11Paths-Date'

AUTHORIZATION_METHOD = '11PATHS'

AUTHORIZATION_HEADER_FIELD_SEPARATOR = ' '

DATE_HEADER_FORMAT = '%Y-%m-%d %H:%M:%S'

X_11PATHS_HEADER_PREFIX = 'X-11paths-'

X_11PATHS_HEADER_SEPARATOR = ':'

property host: str

Host name for Latch service.

property port: int

Port number for Latch service.

property is_https: bool

Whether to use TLS layer or not.

property proxy_host: str | None

Proxy host name.

property proxy_port: int | None

Proxy port number.

classmethod build_paths(*api_version: str*) → *Paths*

Build endpoints dictionary for a given version

Parameters
api_version – Version to get endpoints.

classmethod get_part_from_header(*part: int*, *header: str*) → str

The custom header consists of three parts, the method, the appId and the signature. This method returns the specified part if it exists.

Parameters

- **part** – The zero indexed part to be returned
- **header** – The HTTP header value from which to extract the part

Returns
The specified part from the header or an empty string if not existent

classmethod get_auth_method_from_header(*authorization_header: str*) → str

Get authorization method from header.

Parameters
authorization_header – Authorization HTTP Header

Returns
The Authorization method. Typical values are "Basic", "Digest" or "11PATHS"

classmethod get_app_id_from_header(*authorization_header: str*) → str

Get application identifier from header.

Parameters
authorization_header – Authorization HTTP Header

Returns
The requesting application Id. Identifies the application using the API

classmethod get_signature_from_header(*authorization_header: str*) → str

Get signature from header.

Parameters
authorization_header – Authorization HTTP Header

Returns
The signature of the current request. Verifies the identity of the application using the API

sign_data(*data: str*) → str

Sign data using configured secret.

Parameters
data – the string to sign

Returns
base64 encoding of the HMAC-SHA1 hash of the data parameter using as cipher key.

authentication_headers(
http_method: str,
path_and_query: str,
*,
headers: Mapping[str, str] | None = None,
dt: datetime | None = None,
params: Mapping[str, str] | None = None,
) → dict[str, str]

Calculate the authentication headers to be sent with a request to the API

Parameters

- **http_method** – the HTTP Method, currently only GET is supported
- **path_and_query** – the urlencoded string including the path (from the first forward slash) and the parameters
- **headers** – HTTP headers specific to the 11-paths API. null if not needed.
- **dt** – the Universal Coordinated Time datetime.
- **params** – Request parameters to serialize on body.

Returns

A map with the Authorization and Date headers needed to sign a Latch API request

classmethod build_data_to_sign(
method: str,
dt_str: str,
path_and_query: str,
***,
headers: Mapping[str, str] | None = None,
params: Mapping[str, str] | None = None,
) → str

Build string data to sign.

Parameters

- **method** – HTTP method.
- **dt_str** – Current UTC datetime string.
- **path_and_query** – Path and query string.
- **headers** – Request headers considered (with X-11Path- prefix).
- **params** – Form parameters.

classmethod get_serialized_headers(
headers: Mapping[str, str] | None,
) → str

Prepares and returns a string ready to be signed from the 11-paths specific HTTP headers received

Parameters

headers – a non neccesarily ordered map (array without duplicates) of the HTTP headers to be ordered.

Returns

The serialized headers, an empty string if no headers are passed, or None if there's a problem such as non 11paths specific headers

classmethod get_serialized_params(
params: Mapping[str, str],
) → str

Get paramaters serialized.

Parameters

params – Parameters to serialize.

build_url(*path: str*, *, *query: str | None = None*) → str

Build URL from path and query string.

Parameters

- **path** – URL path
- **query** – Query parameters string

account_pair_with_id(
account_id: str,
*,
web3_account: str | None = None,
web3_signature: str | None = None,
common_name: str | None = None,
phone_number: str | None = None,
) → str

Pairs the origin provider with a user account (mail).

It could raise *warning exceptions* (Recoverable errors) containing the account identifier on exception property *account_id*.

Parameters

- **account_id** – The email for the pairing account - only useful in staging
- **web3_account** – The Ethereum-based account address to pairing the app.
- **web3_signature** – A proof-of-ownership signature with the account address.
- **common_name** – Name send by the service provider attached to this pairing. Typically the identity or name of the user in the SP.
- **phone_number** – Phone number associated with the user.

account_pair(
token: str,
*,
web3_account: str | None = None,
web3_signature: str | None = None,
common_name: str | None = None,
phone_number: str | None = None,
) → str

Pairs the token provider with a user account.

It could raise *warning exceptions* (Recoverable errors) containing the account identifier on exception property *account_id*.

Parameters

- **token** – The token for pairing the app, generated by the Latch mobile app.
- **web3_account** – The Ethereum-based account address to pairing the app.
- **web3_signature** – A proof-of-ownership signature with the account address.
- **common_name** – Name send by the service provider attached to this pairing. Typically the identity or name of the user in the SP.
- **phone_number** – Phone number associated with the user.

account_status(
account_id: str,

*,
instance_id: str | None = None,
operation_id: str | None = None,
silent: bool = False,
nootp: bool = False,
otp_code: str | None = None,
otp_message: str | None = None,
phone_number: str | None = None,
location: str | None = None,
) → *Status*

Return operation status for a given accountId and operation while sending some custom data (Like OTP token or a message).

It could raise `warning exceptions` (Recoverable errors) containing the account status object on *status* exception property.

Parameters

- **account_id** – The account identifier which status is going to be retrieved.
- **instance_id** – The instance identifier.
- **operation_id** – The operation identifier which status is going to be retrieved.
- **silent** – True for not sending lock/unlock push notifications to the mobile devices, false otherwise.
- **nootp** – True for not generating a OTP if needed.
- **otp_code** – Set OTP code manually.
- **otp_message** – Set OTP message.
- **phone_number** – Phone number associated with the user.
- **location** – Allowed location.

account_set_metadata(
account_id: str,
*,
common_name: str | None = None,
phone_number: str | None = None,
) → True

Set account metadata.

Parameters

- **common_name** – Name send by the service provider attached to this pairing. Typically the identity or name of the user in the SP.
- **phone_number** – Phone number associated with the user.

account_unpair(*account_id: str*) → True

Unpairs the origin provider with a user account.

Parameters
account_id – The account identifier.

account_lock(
account_id: str,
*,
instance_id: str | None = None,

operation_id: str | None = None,
) → True

Locks the operation.

Parameters

- **account_id** – The account identifier.
- **instance_id** – The instance identifier.
- **operation_id** – The operation identifier.

account_unlock(
account_id: str,
*,
instance_id: str | None = None,
operation_id: str | None = None,
) → True

Unlocks the operation

Parameters

- **account_id** – The account identifier
- **instance_id** – The instance identifier
- **operation_id** – The operation identifier

account_history(
account_id: str,
*,
from_dt: datetime | None = None,
to_dt: datetime | None = None,
) → *HistoryResponse*

Get history account.

Parameters

- **account_id** – The account identifier.
- **from_dt** – Datetime to start from.
- **to_dt** – Datetime limit.

authorization_control_status(
control_id: str,
*,
location: str | None = None,
send_notifications: bool = True,
) → bool

Get authorization control status.

Parameters

- **control_id** – The Authorization control identifier.
- **location** – Authorized location.
- **send_notification** – Whether to send notification to other participants or not.

operation_create(
parent_id: str,
name: str,

*,
two_factor: ExtraFeature = *ExtraFeature.DISABLED*,
lock_on_request: ExtraFeature = *ExtraFeature.DISABLED*,
) → *Operation*

Add a new operation.

Parameters

- **parent_id** – identifies the parent of the operation to be created.
- **name** – The name of the operation.
- **two_factor** – Specifies if the *Two Factor* protection is enabled for this operation.
- **lock_on_request** – Specifies if the *Lock latches on status request* feature is disabled, opt-in or mandatory for this operation.

operation_update(
operation_id: *str*,
*,
name: *str* | *None* = *None*,
two_factor: ExtraFeature | *None* = *None*,
lock_on_request: ExtraFeature | *None* = *None*,
) → True

Update an operation.

Parameters

- **operation_id** – The operation identifier.
- **name** – The name of the operation.
- **two_factor** – Specifies if the *Two Factor* protection is enabled for this operation.
- **lock_on_request** – Specifies if the *Lock latches on status request* feature is disabled, opt-in or mandatory for this operation.

operation_remove(*operation_id:* *str*) → True

Remove an operation.

Parameters
operation_id – The operation identifier.

operation_list(
*,
parent_id: *str* | *None* = *None*,
) → Iterable[*Operation*]

Get a list of operations.

Parameters
parent_id – To filter by parent operation.

instance_list(
account_id: *str*,
*,
operation_id: *str* | *None* = *None*,
) → Iterable[*Instance*]

Get a list of instances.

Parameters

- **account_id** – The account identifier.

- **operation_id** – The operation identifier.

instance_create(
account_id: str,
name: str,
***,
operation_id: str | None = None,
) → str

Create an instance.

Parameters

- **account_id** – The account identifier.
- **name** – The name of the instance.
- **operation_id** – The operation identifier.

instance_update(
account_id: str,
instance_id: str,
***,
operation_id: str | None = None,
name: str | None = None,
two_factor: ExtraFeature *| None = None*,
lock_on_request: ExtraFeature *| None = None*,
) → True

Update an instance.

Parameters

- **account_id** – The account identifier.
- **instance_id** – The instance identifier.
- **operation_id** – The operation identifier.
- **name** – The name of the instance.
- **two_factor** – Specifies if the *Two Factor* protection is enabled for this instance.
- **lock_on_request** – Specifies if the *Lock latches on status request* feature is disabled, opt-in or mandatory for this instance.

instance_remove(
account_id: str,
instance_id: str,
operation_id: str | None = None,
) → True

Remove the instance.

Parameters

- **account_id** – The account identifier.
- **instance_id** – The instance identifier.
- **operation_id** – The operation identifier.

totp_create(
user_id: str,
common_name: str,
) → *TOTP*

Create a Time-based one-time password.

Parameters

- **user_id** – User identifier (mail)
- **common_name** – Name for the Totp

totp_load(*totp_id: str*) → *TOTP*

Get data information about a given totp.

Parameters
totp_id – Totp Identifier

totp_validate(*totp_id: str*, *code: str*) → True

Validate a code from a totp

Parameters

- **totp_id** – Totp Identifier
- **code** – Code generated

totp_remove(*totp_id: str*) → True

Remove a totp

Parameters
totp_id – Totp Identifier

application_create(
name: str,
*,
two_factor: ExtraFeature = *ExtraFeature.DISABLED*,
lock_on_request: ExtraFeature = *ExtraFeature.DISABLED*,
contact_phone: str = '',
contact_mail: str = '',
) → *ApplicationCreateResponse*

Create a new application

Parameters

- **name** – Name of new application
- **two_factor** – Specifies if the *Two Factor* protection is enabled for this instance.
- **lock_on_request** – Specifies if the *Lock latches on status request* feature is disabled, opt-in or mandatory for this instance.
- **contact_phone** – Contact phone number.
- **contact_mail** – Contact email.

subscription_load() → *UserSubscription*

Returns the developer's subscription.

application_list() → Iterable[*Application*]

Returns the list of registered applications.

application_update(
application_id: str,
*,
name: str | *None* = *None*,

two_factor: ExtraFeature | *None* = *None*,
lock_on_request: ExtraFeature | *None* = *None*,
contact_phone: *str* | *None* = *None*,
contact_mail: *str* | *None* = *None*,
) → True

Modify a given application

Parameters

- **application_id** – Application identifier
- **name** – Name of new application
- **two_factor** – Specifies if the *Two Factor* protection is enabled for this instance.
- **lock_on_request** – Specifies if the *Lock latches on status request* feature is disabled, opt-in or mandatory for this instance.
- **contact_phone** – Contact phone number.
- **contact_mail** – Contact email.

application_remove(*application_id:* *str*) → True

Remove a given application.

Parameters
application_id – Application identifier

latch_sdk.sansio.**TFactory**

Factory type alias.

alias of Callable[[*Response*], *TReturnType*]

class latch_sdk.sansio.**LatchSDKSansIO**(*core:* LatchSansIO[TResponse])

Bases: Generic[*TResponse*]

Latch SDK sans-io.

Parameters
core – Latch core driver.

property core: *LatchSansIO*[*TResponse*]

Latch core driver.

latch_sdk.sansio.**response_no_error**(*resp:* Response) → True

Returns
True if no error.

latch_sdk.sansio.**response_account_pair**(*resp:* Response) → str

Returns
Account identifier.

latch_sdk.sansio.**response_account_status**(
resp: Response,
) → *Status*

Returns
Status object.

latch_sdk.sansio.**response_account_history**(
resp: Response,
) → *HistoryResponse*

Returns
History response object.

latch_sdk.sansio.**response_authorization_control_status**(*resp:* Response) → bool

Returns
False if authorization control is locked, and True otherwise.

latch_sdk.sansio.**response_operation**(
resp: Response,
) → *Operation*

Returns
Operation object.

latch_sdk.sansio.**response_operation_list**(
resp: Response,
) → Iterable[*Operation*]

Returns
Operation object list.

latch_sdk.sansio.**response_instance_list**(
resp: Response,
) → Iterable[*Instance*]

Returns
Instance object list.

latch_sdk.sansio.**response_instance_create**(*resp:* Response) → str

Returns
Instance identifier.

latch_sdk.sansio.**response_totp**(*resp:* Response) → *TOTP*

Returns
TOTP object.

latch_sdk.sansio.**response_subscription**(
resp: Response,
) → *UserSubscription*

Returns
Developer's subscription.

latch_sdk.sansio.**response_application_create**(
resp: Response,
) → *ApplicationCreateResponse*

Returns
Application creation information.

latch_sdk.sansio.**response_application_list**(
resp: Response,
) → Iterable[*Application*]

Returns
Application object list.

CHAPTER

THREE

COMMAND-LINE INTERFACE APPLICATION

Latch SDK provides a command-line interface application to manage your Latch applications.

Note

In order to be able to use the command-lina application an extra requirements must be installed.

```
$ pip install latch-sdk-telefonica[cli]
```

Contents

- *remove*
- *update*

* *authorization-control*
  - *status*
* *operation*
  - *create*
  - *list*
  - *remove*
  - *update*
* *subscription*
* *totp*
  - *create*
  - *load*
  - *remove*
  - *validate*

3.1 latchcli

Latch command-line application

```
latchcli [OPTIONS] COMMAND [ARGS]...
```

Options

--version

Show the version and exit.

-a, --app-id <app_id>

Required Application or user identifier

-s, --secret <secret>

Required Secret or password

Environment variables

LATCH_APP_ID

Provide a default for *--app-id*

LATCH_SECRET

Provide a default for *--secret*

3.1.1 account

Account actions

```
latchcli account [OPTIONS] COMMAND [ARGS]...
```

history

Show latch actions history

```
latchcli account history [OPTIONS] ACCOUNT_ID
```

Options

--from <from_dt>

Date time from

--to <to_dt>

Date time to

Arguments

ACCOUNT_ID

Required argument

instance

Instances actions

```
latchcli account instance [OPTIONS] ACCOUNT_ID COMMAND [ARGS]...
```

Options

-o, --operation-id <operation_id>

Operation identifier

Arguments

ACCOUNT_ID

Required argument

create

Create a new instance

```
latchcli account instance ACCOUNT_ID create [OPTIONS]
```

Options

-n, --name <name>

Required Intance name

list

Print available instances list

```
latchcli account instance ACCOUNT_ID list [OPTIONS]
```

remove

Remove a given instance

```
latchcli account instance ACCOUNT_ID remove [OPTIONS] INSTANCE_ID
```

Arguments

INSTANCE_ID

Required argument

update

Update a given instance

```
latchcli account instance ACCOUNT_ID update [OPTIONS] INSTANCE_ID
```

Options

-n, --name <name>

Operation name

-t, --two-factor <two_factor>

Two factor feature

Options

MANDATORY | OPT_IN | DISABLED

-l, --lock-on-request <lock_on_request>

Lock on request feature

Options

MANDATORY | OPT_IN | DISABLED

Arguments

INSTANCE_ID

Required argument

lock

Lock a latch

```
latchcli account lock [OPTIONS] ACCOUNT_ID
```

Options

-o, --operation-id <operation_id>
Operation identifier

-i, --instance-id <instance_id>
Instances identifier

Arguments

ACCOUNT_ID
Required argument

pair

Pair a new latch.

```
latchcli account pair [OPTIONS] TOKEN
```

Options

-a, --web3-account <web3_account>
Web3 account identifier

-s, --web3-signature <web3_signature>
Web3 signature

-n, --common-name <common_name>
Common name

-p, --phone-number <phone_number>
Phone number

Arguments

TOKEN
Required argument

pair-with-id

Pair with user id a new latch.

```
latchcli account pair-with-id [OPTIONS] USER_ID
```

Options

-a, --web3-account <web3_account>
Web3 account identifier

-s, --web3-signature <web3_signature>
Web3 signature

-n, --common-name <common_name>
Common name

-p, --phone-number <phone_number>
Phone number

Arguments

USER_ID

Required argument

set-metadata

Update account metadata

```
latchcli account set-metadata [OPTIONS] ACCOUNT_ID
```

Options

-n, --common-name <common_name>

Common name

-p, --phone-number <phone_number>

Phone number

Arguments

ACCOUNT_ID

Required argument

status

Get latch status

```
latchcli account status [OPTIONS] ACCOUNT_ID
```

Options

-o, --operation-id <operation_id>

Operation identifier

-i, --instance-id <instance_id>

Instances identifier

--nootp

Avoid OTP

--silent

Do not push notification

-c, --otp-code <otp_code>

Set OTP code

-m, --otp-message <otp_message>

Set OTP message

-p, --phone-number <phone_number>

Phone number

-l, --location <location>

Location identifier

Arguments

ACCOUNT_ID

Required argument

unlock

Unlock a latch

```
latchcli account unlock [OPTIONS] ACCOUNT_ID
```

Options

-o, --operation-id <operation_id>

Operation identifier

-i, --instance-id <instance_id>

Instances identifier

Arguments

ACCOUNT_ID

Required argument

unpair

Unpair a latch

```
latchcli account unpair [OPTIONS] ACCOUNT_ID
```

Arguments

ACCOUNT_ID

Required argument

3.1.2 application

Application actions

```
latchcli application [OPTIONS] COMMAND [ARGS]...
```

create

Create a new application

```
latchcli application create [OPTIONS]
```

Options

-n, --name <name>

Required Operation name

-t, --two-factor <two_factor>

Two factor feature

Options

MANDATORY | OPT_IN | DISABLED

-l, --lock-on-request <lock_on_request>

Lock on request feature

Options

MANDATORY | OPT_IN | DISABLED

-m, --contact-mail <contact_mail>

Conctact email

-p, --contact-phone <contact_phone>

Conctact phone number

list

List registered applications

```
latchcli application list [OPTIONS]
```

remove

Remove a given application

```
latchcli application remove [OPTIONS] APPLICATION_ID
```

Arguments

APPLICATION_ID

Required argument

update

Update a given application

```
latchcli application update [OPTIONS] APPLICATION_ID
```

Options

-n, --name <name>

Operation name

-t, --two-factor <two_factor>

Two factor feature

Options

MANDATORY | OPT_IN | DISABLED

-l, --lock-on-request <lock_on_request>

Lock on request feature

Options

MANDATORY | OPT_IN | DISABLED

-m, --contact-mail <contact_mail>
Conctact email

-p, --contact-phone <contact_phone>
Conctact phone number

Arguments

APPLICATION_ID
Required argument

3.1.3 authorization-control

Authorization Control actions

```
latchcli authorization-control [OPTIONS] COMMAND [ARGS]...
```

status

Get authorization control status

```
latchcli authorization-control status [OPTIONS] CONTROL_ID
```

Options

--silent
Do not push notification

-l, --location <location>
Location identifier

Arguments

CONTROL_ID
Required argument

3.1.4 operation

Operations actions

```
latchcli operation [OPTIONS] COMMAND [ARGS]...
```

create

Create a new operation

```
latchcli operation create [OPTIONS]
```

Options

-p, --parent-id <parent_id>
Required Parent operation or application identifier

-n, --name <name>

Required Operation name

-t, --two-factor <two_factor>

Two factor feature

Options

MANDATORY | OPT_IN | DISABLED

-l, --lock-on-request <lock_on_request>

Lock on request feature

Options

MANDATORY | OPT_IN | DISABLED

list

Print available operations list

```
latchcli operation list [OPTIONS]
```

Options

-p, --parent-id <parent_id>

Parent operation identifier

remove

Remove a given operation

```
latchcli operation remove [OPTIONS] OPERATION_ID
```

Arguments

OPERATION_ID

Required argument

update

Update a given operation

```
latchcli operation update [OPTIONS] OPERATION_ID
```

Options

-n, --name <name>

Operation name

-t, --two-factor <two_factor>

Two factor feature

Options

MANDATORY | OPT_IN | DISABLED

-l, --lock-on-request <lock_on_request>

Lock on request feature

Options

MANDATORY | OPT_IN | DISABLED

Arguments

OPERATION_ID

Required argument

3.1.5 subscription

Retrieve developer's subscription information.

```
latchcli subscription [OPTIONS]
```

3.1.6 totp

TOTP actions

```
latchcli totp [OPTIONS] COMMAND [ARGS]...
```

create

Create a new TOTP

```
latchcli totp create [OPTIONS]
```

Options

-u, --user-id <user_id>

Required User identifier

-n, --common-name <common_name>

Required Common name

load

Load a given TOTP

```
latchcli totp load [OPTIONS] TOTP_ID
```

Arguments

TOTP_ID

Required argument

remove

Remove a given TOTP

```
latchcli totp remove [OPTIONS] TOTP_ID
```

Arguments

TOTP_ID

Required argument

validate

Validate TOTP code

```
latchcli totp validate [OPTIONS] TOTP_ID
```

Options

-c, --code <code>

Required Temporal code

Arguments

TOTP_ID

Required argument

CHAPTER

FOUR

RELEASE AND BRANCHING MODEL

The default branch is *main*. There are two branches types protected against pushing commits directly: *main* and *release/X.Y* branches. Any new code pushed to protected branches must be done by pull requests.

The library version is defined on *pyproject.toml* at section *project* and key *version*. It uses PEP440 format and Semver 2.0.0 meaning. It must contain the **next stable version** to release. In the case of *main* branch, the *patch* part must be *0*. Because the *patch* part only increase when there is a bug fix in already released version. On the other hand, in *release/1.7* branch version could be *1.7.1* because the version *1.7.0* has been released previously.

Each time new code is pushed to *main* branch (by pull request) it will create a beta release with current code. For example, if we are developing the future release *1.7.0* it will create a beta release like *1.7.0b4*.

When the code is considered stable and it is going to be released. We must create a branch *release/1.7* from *main*. Automatically it will create a new pull request to *main* bumping verision to *1.8.0*. Any new code pushed to *release/X.Y* branches will create a release candidate release like *1.7.0rc3*. When a release candidate version is considered ready to release, a manual action will be lauched by maintainers in order to create the final release *1.7.0*. It will create automatically a pull request to *release/1.7* bumping version to *1.7.1*.

It is posible to generate a release from any branch. For any branch, except regular ones, it will create an alpha version.

Branch	Version level	Action on push
main	beta	Create release
release/X.Y	release candidate	Create release and bump minor version part on develop branch if needed
any	alpha	None

Important

NEVER CHANGE THE VERSION MANUALLY

There is an action to bump version on develop if you want to increase the *major* part.

Any new release created will be build and published on PyPi. Even pre-release versions. So, anybody could donwload an alpha or beta version under their own risk.

CHAPTER

FIVE

DEVELOPING SDK

5.1 Requirements

- Python >=*3.9*
- Poetry >=*2.0.0*

5.2 Setting up the repository

The first step is to generate a virtual environment with poetry:

```
$ poetry install --all-extras
```

It will create a venv for your default python version and install all requirements and extras.

5.3 Code style

We use ruff as code style formatter and checker and isort and absolufy-imports (deprecated, looking for alternative) to format and check imports.

In order to check your code you can run:

```
$ make lint
```

But, if you want to ensure your code is following rules you can reformat it using:

```
$ make beautify
```

5.4 Testing

All tests must be in *tests/* directory. We use pytest as test runner.

You could run all tests and coverage analysis using:

```
$ make tests
```

PYTHON MODULE INDEX

l

INDEX

Symbols

A

B

C

D

E

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X