Intellifire4Py

Jeff Stein

Feb 24, 2023

CONTENTS

CHAPTER

ONE

FEATURES

- TODO

CHAPTER

TWO

REQUIREMENTS

- TODO

CHAPTER

THREE

INSTALLATION

You can install *Intellifire4Py* via pip from PyPI:

```
$ pip install intellifire4py
```

CHAPTER

FOUR

USAGE

Please see the *API Reference* for details.

CHAPTER

FIVE

CONTRIBUTING

Contributions are very welcome. To learn more, see the *Contributor Guide*.

CHAPTER

SIX

LICENSE

Distributed under the terms of the *MIT license*, *Intellifire4Py* is free and open source software.

CHAPTER

SEVEN

ISSUES

If you encounter any problems, please file an issue along with a detailed description.

CHAPTER

EIGHT

CREDITS

This project was generated from @cjolowicz's Hypermodern Python Cookiecutter template.

8.1 Usage

There are two main APIs in version 3.0 of IntelliFire4Py

8.1.1 Control Config Values

In order to actually issue commands to the fireplace you will need to obtain a few items from the cloud portal. These can be done automatically

- `user_id` - This is a the `user_id` associated with your specific account
- `api_key` - This is a specific key associated with a specific fireplace
- `fireplace_ip` - The IP address of the fireplace on the local network

8.1.2 UDP Discovery

This code is also available in `example_discovery.py`:

```
import asyncio
from intellifire4py.udp import UDPFireplaceFinder

async def main() -> None:
    """Discovery fire places"""

    # Most likely fail discovery due to a short time out
    timeout = 1
    print(f"----- Find Fire Places - (waiting {timeout} seconds)-----")
    af = UDPFireplaceFinder()
    print(await af.search_fireplace(timeout=timeout))

    # Set a reasonalbe timeout
    print(f"----- Find Fire Places - (waiting {timeout} seconds)-----")
    af = UDPFireplaceFinder()
    print(await af.search_fireplace(timeout=timeout))
```

(continues on next page)

(continued from previous page)

```
if __name__ == "__main__":
    loop = asyncio.get_event_loop()
    loop.run_until_complete(main())
```

8.1.3 Local Polling

With only access to the ip address of the unit you can perform local polling of data using `IntelliFireAPILocal`.

```
import asyncio
import logging
import os

from intellifire4py import IntelliFireAPILocal

logging.basicConfig(level=logging.DEBUG)

async def main() -> None:
    """Main function."""
    print(
        """
    Accessing IFT_IP environment variable to connect to fireplace
    """
    )
    ip = os.environ["IFT_IP"]

    api = IntelliFireAPILocal(fireplace_ip=ip)
    await api.poll(suppress_warnings=False)
    print(api.data)


if __name__ == "__main__":
    loop = asyncio.get_event_loop()
    loop.run_until_complete(main())
```

8.1.4 Cloud Credentials

In order to actually control the unit you will need to access the cloud in order to pull down some credentials. This is demonstrated in `example_cloud_info.py` however the key usage is as follows:

```
cloud_api = IntelliFireAPICloud(use_http=True, verify_ssl=False)
await cloud_api.login(username=username, password=password)

# Once logged in you can pull out the api key for the default (first detected) fireplace
api_key = cloud_api.get_fireplace_api_key(cloud_api.default_fireplace)

# Extract user_id
user_id = cloud_api.get_user_id()
```

When obtained these values can then be used for local control of the fireplace

8.1.5 Local Control

In order to control the fireplace you must instantiate `IntelliFireAPILocal` as follows:

```
from intellifire4py import IntelliFireAPILocal

api = IntelliFireAPILocal(
    fireplace_ip=fireplace_ip,
    user_id=user_id,
    api_key=api_key
)

# And then you can issue commands such as:
await api.flame_on()
```

8.2 Reference

8.2.1 IntelliFire4PY

IntelliFire API for Python.

8.2.2 IntelliFireAPILocal

class intellifire4py.**IntelliFireAPILocal**(*fireplace_ip*, *user_id=''*, *api_key=''*)

Top level API for IntelliFire Data - local network only.

Class Initialization.

Parameters

- **fireplace_ip** (*str*) – _description_
- **user_id** (*str, optional*) – The *user_id* as retrieved from *IntelliFireAPICloud*. If left blank - will not be able to control the unit. Defaults to “”.
- **api_key** (*str, optional*) – Each fireplace has a unique *api_key*. If left blank - will not be able to control the unit. Defaults to “”.

See also:

- *IntelliFireAPICloud.login()*
- *IntelliFireAPICloud.get_user_id()*
- *IntelliFireAPICloud.get_fireplace_api_key()*

async beep()

Issue a beep command (Cloud Only).

Return type
None

property data: IntelliFirePollData

Return data to the user.

async fan_off()

Turn fan off.

Return type
None

async flame_off()

Turn off the flame.

Return type
None

async flame_on()

Turn on the flame.

Return type
None

property is_polling_in_background: bool

Return whether api is polling.

log_status()

Log a status message.

Return type
None

overwrite_data(*new_data*)

Overwrite existing poll data.

Parameters
new_data (*IntelliFirePollData*) –

Return type
None

async pilot_off()

Turn off the pilot light.

Return type
None

async pilot_on()

Turn on the pilot light.

Return type
None

async poll(*suppress_warnings=False*)

Read the /poll endpoint.

Parameters
suppress_warnings (*bool, optional*) – If *True* will inhibit the printing of log messages Useful for a specific case in Home Assistant. Defaults to False.

Raises
ConnectionError – _description_

Return type
None

async send_command(*, *command*, *value*)

Send a command (local only for now).

Parameters

- **command** (*IntelliFireCommand*) –
- **value** (*int*) –

Return type
None

async set_fan_speed(*speed*)

Set fan speed.

Parameters
speed (*int*) –

Return type
None

async set_flame_height(*height*)

Set flame height.

Parameters
height (*int*) – Valid height *0-4* (in the future this will be 1-5)

Return type
None

async set_lights(*level*)

Modify light levels.

Parameters
level (*int*) –

Return type
None

async set_sleep_timer(*minutes*)

Set the sleep timer in minutes.

Parameters
minutes (*int*) – Valid range *0-180*

Return type
None

async set_thermostat_c(*temp_c*)

Set thermostat value in centigrade.

Parameters
temp_c (*int*) –

Return type
None

async set_thermostat_f(*temp_f*)

Set thermostat value in fahrenheit.

Example

```
# Set to 70 and store the value internally
await ift_control.set_thermostat_f(temp_f=70)
# Turn off thermostat
await ift_control.turn_off_thermostat()
# Turn on thermostat - will remember the last temp (70)
await ift_control.turn_on_thermostat()
```

Parameters
temp_f (*int*) –

Return type
None

async soft_reset()

Issue a soft reset command (Cloud Only).

Return type
None

async start_background_polling(*minimum_wait_in_seconds=15*)

Start an ensure-future background polling loop.

Parameters
minimum_wait_in_seconds (*int*) –

Return type
None

async stop_background_polling()

Stop background polling - return whether it had been polling.

Return type
bool

async stop_sleep_timer()

Stop the sleep timer.

Return type
None

async turn_off_thermostat()

Turn off thermostat mode.

Return type
None

async turn_on_thermostat()

Turn on thermostat mode.

Return type
None

8.2.3 IntelliFireAPICloud

class intellifire4py.**IntelliFireAPICloud**(*, *use_http=False*, *verify_ssl=True*)

Api for cloud access.

Initialize the class.

In most cases you should not specify either the *use_http* or *verify_ssl* parameters - however in some special cases such as protected networks you may need these options.

Parameters

- **use_http** (*bool, optional*) – whether to use HTTP or HTTPS mode. Defaults to False.
- **verify_ssl** (*bool, optional*) – Enable/Disable SSL Verification. Defaults to True.

async **beep**()

Issue a beep command (Cloud Only).

Return type
None

property **data: IntelliFirePollData**

Return data to the user.

async **fan_off**()

Turn fan off.

Return type
None

async **flame_off**()

Turn off the flame.

Return type
None

async **flame_on**()

Turn on the flame.

Return type
None

get_fireplace_api_key(*fireplace=None*)

Get API key for specific fireplace.

Parameters
fireplace (*IntelliFireFireplace | None*) –

Return type
str

async **get_fireplaces**(*client*, *, *location_id*)

Get fireplaces at a location with associated API keys!.

Parameters

- **client** (*AsyncClient*) –
- **location_id** (*str*) –

Return type
list[intellifire4py.model.IntelliFireFireplace]

async get_locations(*client*)

Enumerate configured locations that a user has access to.

'location_id' can be used to discovery fireplaces and associated serial numbers + api keys at a give location.

Parameters
client (*AsyncClient*) –

Return type
list[dict[str, str]]

get_user_id()

Get user ID from cloud.

Return type
str

property is_polling_in_background: bool

Return whether api is polling.

async login(*, *username*, *password*)

Login to Cloud API.

Parameters

- **username** (*str*) – IFTAPI.net Username (usually email)
- **password** (*str*) – IFTAPI.net Password

Raises
LoginError – _description_

Returns
None

Return type
None

async long_poll(*fireplace=None*)

Perform a LongPoll to wait for a Status update.

Only returns a status update when the fireplace's status actually changes (excluding normal periodic decreases in the "time remaining" field). If the fireplace status does not change during the time period, the server returns status code *408* after the time limit is exceeded. The app can then immediately issue another request on this function. If the status changes, then the server returns a *200* status code, the status content (in the same format as for apppoll), and an Etag header. The Etag should be sent in an If-None- Match header for the next request, so the server knows where in the queue to look for the next command to return. The correct order to do this is first issue an apppoll request (or equivalently, an enumuserfireplaces request), and then issue applongpoll requests for as long as the status is needed. Although this may seem to create a race condition, the server puts fireplace status updates in a queue where they last for *30* seconds. Therefore, as long as the Internet connection isn't unusably slow, no status updates will be lost. If the connection goes down, then the process needs to be restarted. The time limit is nominally *60* seconds. After *57* seconds, the server will send a *408* response, and after *61* seconds, the mobile app should assume that the connection has been dropped.

Parameters
fireplace (*IntelliFireFireplace | None, optional*) – _description_. Defaults to None.

Raises
ApiCallError – Issue with the API call, either bad credentials or a bad serial number

Returns
True if status changed, *False* if it did not

Return type
bool

`overwrite_data`(*new_data*)

Overwrite existing poll data.

Parameters
new_data (*IntelliFirePollData*) –

Return type
None

`async pilot_off`()

Turn off the pilot light.

Return type
None

`async pilot_on`()

Turn on the pilot light.

Return type
None

`async poll`(*fireplace=None*)

Return a fireplace's status in JSON.

Parameters
fireplace (*IntelliFireFireplace | None, optional*) – _description_. Defaults to None.

Raises

- **ApiCallError** – _description_
- **ApiCallError** – _description_
- **Exception** – _description_

Returns
description

Return type
type

Example:

```
{
"name":"undefined",
"temperature":"22",
"battery":"0",
"pilot":"0",
"light":"3",
"height":"4",
"fanspeed":"0",
"hot":"0",
"power":"0",
"schedule_enable":"0",
```

(continues on next page)

(continued from previous page)


```
"thermostat":"0",
"setpoint":"0",
"timer":"0",
"timeremaining":"0",
"prepurge":"0",
"feature_light":"1",
"feature_thermostat":"1",
"power_vent":"0",
"feature_fan":"1",
"errors":[3269],
"firmware_version":"0x01000000"
"brand":"H&G"
}
```


async send_command(*, *command*, *value*)

Send a command (cloud based).

Parameters

- **command** (*IntelliFireCommand*) –
- **value** (*int*) –

Return type
None

async set_fan_speed(*speed*)

Set fan speed.

Parameters
speed (*int*) –

Return type
None

async set_flame_height(*height*)

Set flame height.

Parameters
height (*int*) – Valid height *0-4* (in the future this will be 1-5)

Return type
None

async set_lights(*level*)

Modify light levels.

Parameters
level (*int*) –

Return type
None

async set_sleep_timer(*minutes*)

Set the sleep timer in minutes.

Parameters
minutes (*int*) – Valid range *0-180*

Return type
None

async set_thermostat_c(*temp_c*)

Set thermostat value in centigrade.

Parameters
temp_c (*int*) –

Return type
None

async set_thermostat_f(*temp_f*)

Set thermostat value in fahrenheit.

Example

```
# Set to 70 and store the value internally
await ift_control.set_thermostat_f(temp_f=70)
# Turn off thermostat
await ift_control.turn_off_thermostat()
# Turn on thermostat - will remember the last temp (70)
await ift_control.turn_on_thermostat()
```

Parameters
temp_f (*int*) –

Return type
None

async soft_reset()

Issue a soft reset command (Cloud Only).

Return type
None

async start_background_polling(*minimum_wait_in_seconds=10*)

Start an ensure-future background polling loop.

Parameters
minimum_wait_in_seconds (*int*) –

Return type
None

async stop_background_polling()

Stop background polling - return whether it had been polling.

Return type
bool

async stop_sleep_timer()

Stop the sleep timer.

Return type
None

async turn_off_thermostat()

Turn off thermostat mode.

Return type
None

async turn_on_thermostat()

Turn on thermostat mode.

Return type
None

8.2.4 IntelliFireErrorCode

class intellifire4py.**IntelliFireErrorCode**(*value=<no_arg>*, *names=None*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

The following is a description of various error codes. These were obtained by decompiling the Android APK.

PILOT_FLAME

Pilot Flame Error: Your appliance has been safely disabled. Please contact your dealer and report this issue.

FAN_DELAY

Fan Information: Fan will turn on within 3 minutes. Your appliance has a built-in delay that prevents the fan from operating within the first 3 minutes of turning on the appliance. This allows the air to be heated prior to circulation.

FLAME

Pilot Flame Error. Your appliance has been safely disabled. Please contact your dealer and report this issue.

MAINTENANCE

Maintenance: Your appliance is due for a routine maintenance check. Please contact your dealer to ensure your appliance is operating at peak performance.

DISABLED

Appliance Safely Disabled: Your appliance has been disabled. Please contact your dealer and report this issue.

FAN

Fan Error. Your appliance has detected that an accessory is not functional. Please contact your dealer and report this issue.

LIGHTS

Lights Error. Your appliance has detected that an accessory is not functional. Please contact your dealer and report this issue.

ACCESSORY

Your appliance has detected that an AUX port or accessory is not functional. Please contact your dealer and report this issue.

SOFT_LOCK_OUT

Sorry your appliance did not start. Try again by pressing Flame ON.

OFFLINE

Your appliance is currently offline.

ECM_OFFLINE

ECM is offline.

8.3 Contributor Guide

Thank you for your interest in improving this project. This project is open-source under the MIT license and welcomes contributions in the form of bug reports, feature requests, and pull requests.

Here is a list of important resources for contributors:

- Source Code
- Documentation
- Issue Tracker
- *Code of Conduct*

8.3.1 How to report a bug

Report bugs on the Issue Tracker.

When filing an issue, make sure to answer these questions:

- Which operating system and Python version are you using?
- Which version of this project are you using?
- What did you do?
- What did you expect to see?
- What did you see instead?

The best way to get your bug fixed is to provide a test case, and/or steps to reproduce the issue.

8.3.2 How to request a feature

Request features on the Issue Tracker.

8.3.3 How to set up your development environment

You need Python 3.7+ and the following tools:

- Poetry
- Nox
- nox-poetry

Install the package with development requirements:

```
$ poetry install
```

You can now run an interactive Python session, or the command-line interface:

```
$ poetry run python
$ poetry run intellifire4py
```

8.3.4 How to test the project

Run the full test suite:

```
$ nox
```

List the available Nox sessions:

```
$ nox --list-sessions
```

You can also run a specific Nox session. For example, invoke the unit test suite like this:

```
$ nox --session=tests
```

Unit tests are located in the *tests* directory, and are written using the pytest testing framework.

8.3.5 How to submit changes

Open a pull request to submit changes to this project.

Your pull request needs to meet the following guidelines for acceptance:

- The Nox test suite must pass without errors and warnings.
- Include unit tests. This project maintains 100% code coverage.
- If your changes add functionality, update the documentation accordingly.

Feel free to submit early, though—we can always iterate on this.

To run linting and code formatting checks before committing your change, you can install pre-commit as a Git hook by running the following command:

```
$ nox --session=pre-commit -- install
```

It is recommended to open an issue before starting work on anything. This will allow a chance to talk it over with the owners and validate your approach.

8.4 Contributor Covenant Code of Conduct

8.4.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, caste, color, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

8.4.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

8.4.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

8.4.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

8.4.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at jeffstein@gmail.com. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

8.4.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

8.4.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.1, available at https://www.contributor-covenant.org/version/2/1/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

8.5 License

PYTHON MODULE INDEX

INDEX

S

T