iamai

Release 0.0.3rc3

Hsiang Nianian

Feb 23, 2024

CONTENTS

Welcome to iamai, a powerful and comprehensive AI toolkit that seamlessly integrates multimodal machine learning capabilities with advanced tools for cross-platform robot development!

> This library is designed to provide developers with a unified solution for creating intelligent systems that span multiple modalities and operate across diverse platforms.

- Interactive docs & demos
- Seamless migration: Works for both Rasa and GPT and more…
- Fully tree shakeable: Only take what you want, bundle size
- Flexible: Configurable event filters and targets
- Optional Add-ons: Apscheduler, etc.
- Cross-platform: dingtalk etc.

Iamai is not just a library; it's a comprehensive AI toolkit that brings together multimodal machine learning and cross-platform robotics. Whether you're developing intelligent systems or constructing robots for various platforms, iamai is your go-to solution for a unified and powerful development experience. Explore the possibilities with iamai today!

This documentation is based on Alicebot Docs, but with some modifications and improvements. It still has many shortcomings and is currently under reconstruction.

CHAPTER

ONE

QUICK START

1.1 Installation

Tip: iamai only supports Python 3.8+ versions.

Install using the Python package manager (pip):

```
pip install iamai
```

Install the latest development version from GitHub.

```
git clone https://github.com/retrofor/iamai.git
cd iamai
poetry install --no-dev # Recommended
```

1.1.1 Installing Adapters

iamai itself is just a chatbot framework and requires additional installation of adapters for supporting specific protocols.

You can use pip to install protcol adapters

```
pip install iamai-adapter-cqhttp
pip install iamai-adapter-mirai
pip install iamai-adapter-dingtalk
```

Alternatively, you can install iamai along with the corresponding adapters at the same time, like this

```
pip install iamai[all]
pip install iamai[onebot11]
pip install iamai[dingtalk]
```

1.1.2 First Project

This section will guide you to build a simple iamai bot project from scratch.

1. Create and enter a new directory

   mkdir iamai-starter && cd iamai-starter

2. Create a *main.py* file and write the following content.

```python
from iamai import Bot

bot = Bot()

if __name__ == "__main__":
  bot.run()
```

3. Create a *config.toml* file and write the following content inside that file:

```toml
[bot]
plugin_dirs = ["plugins"]
adapters = ["iamai.adapter.cqhttp"]
```

4. Create a *plugins* directory

   mkdir plugins

5. Try Running *main.py*

   python main.py

You should see the following output log

```
2021-07-24 00:00:00.000 | INFO | iamai.bot:_load_plugins_from_dirs:689 - Loading plugins␣
↪from dirs "/xxx/plugins"
2021-07-24 00:00:00.000 | INFO | iamai.bot:_load_adapters:746 - Succeeded to load␣
↪adapter "Onebot11Adapter" from "iamai.adapter.onebot11"
2021-07-24 00:00:00.000 | INFO | iamai:run:90 - Running iamai...
```

1.1.3 Directory Structure

iamai recommends the following directory structure:

<FileTree>

<FileTree.Folder name="plugins (The plugins dir)" defaultOpen>
<FileTree.File name="xxx.py" />

</FileTree.Folder> <FileTree.File name="config.toml (The configuration file)" /> <FileTree.File name="main.py" />

</FileTree>

The *main.py* and *config.toml* files are as shown above.

1.1.4 Configuring the Protocol Enpoint

The above example uses the *iamai.adapter.cqhttp* protocol adapter, which is an adapter for the OneBot v11 protocol (formerly known as the CKYU platform's CQHTTP protocol). It requires a protocol endpoint compatible with the OneBot protocol for communication. Here are some commonly used QQ protocol endpoints that support the OneBot protocol:

- [go-cqhttp](https://github.com/Mrs4s/go-cqhttp)
- [mirai](https://github.com/mamoe/mirai) + [onebot-kotlin](https://github.com/yyuueexxiinngg/onebot-kotlin)
- [oicq](https://github.com/takayama-lily/oicq)

For more information, see the [CQHTTP Protocol Usage Guide](./cqhttp-adapter.md) .

You can also install other protocols adapters or try writing your own protocol adapter.

Development Tips

When developing with iamai, it is recommended to use an IDE with type checking, such as PyCharm, VSCode, etc. This can help you make full use of iamai's type hints.

CHAPTER

TWO

BASIC CONFIGURATION

2.1 Configuration File

All configurations for iamai are stored in `config.toml` file.

The `config.toml` file is a standard TOML v1.0.0 file. TOML is a 'minimal' configuration file format that easy to read due to obvious semantics. It is recommended to have a basic understanding of the TOML language before proceeding.

iamai configurations are stored in different tables within the config.toml file. iamai's own configurations are in the `bot` table, while all `adapter` and `plugin` configurations are stored in the `adapter` and `plugin` tables, respectively.

iamai iteself has the following configurations:

- **plugins** The list of plugins to be loaded , which will be loaded using the `Bot` class's `load_plugins()` method.
- **plugin_dirs** The list of plugin directories to be loaded , which will be loaded using the `Bot` classes `load_plugins_from_dirs()` method.
- **adapters** The list of adapters to be loaded, which will be loaded sequentially using the `Bot` classes `load_adapters()` method.

Logging-related configurations are found in the `bot.log` table, as follows:

- **level** The log level.
- **verbose_exception** Detailed exception logging. When set to True, it will add the exception's Tracebackto the log.

Configurations for different adapters or plugins will be placed in sub-tables under `adapter` and `plugin`.

For example, a configuration file with the `cqhttp` adapter configuration looks like this:

```
[bot]
# iamai's own configurations
plugins = []
plugin_dirs = ["plugins"]
adapters = ["iamai.adapter.cqhttp"]

[bot.log]
# Logging-related configurations
level = "INFO"
verbose_exception = true

[adapter.cqhttp]
# Configuration for the CQHTTP adapter
adapter_type = "reverse-ws"
```

(continues on next page)

(continued from previous page)

```
host = "127.0.0.1"
port = 8080
url = "/cqhttp/ws"
```

You can include any custom, undefined configuration options, and they will be loaded by iamai. These custom configurations can be used in plugins. For example, you can define `superuser` to represent a special user controlling the current machine, or use `nickname` to represent the nickname of the current bot.

```
# Custom, unused keys
superuser = 10001
nickname = "Little AI"

[bot]
# iamai's own configurations
plugins = []
plugin_dirs = ["plugins"]
adapters = ["iamai.adapter.cqhttp"]
```

In a plugin, you can access the entire configuration using `self.bot.config`.For example:

```
from iamai import Plugin


class Halloiamai(Plugin):
    async def handle(self) -> None:
        await self.event.reply(f"Hello, I am {self.bot.config.nickname}!")

    async def rule(self) -> bool:
        if self.event.adapter.name != "cqhttp":
            return False
        if self.event.type != "message":
            return False
        return (
            self.event.user_id == self.bot.config.superuser
            and str(self.event.message).lower() == "hello"
        )
```

2.1.1 Custom Configuration File or No Configuration File

You can provide a custom configuration file or use no configuration file by passing the config_file or config_dict attributes when instantiating the Bot object.

iamai will check the file extension of config_file, allowing either .toml or .json files. If the configuration file is a JSON file, it should be a standard JSON file encoded in UTF-8, with content equivalent to the TOML format configuration file.

When specifying the config_dict attribute, iamai will no longer read from the configuration file and will directly read from the given configuration dictionary.

```
# Custom configuration file name
```

(continues on next page)

(continued from previous page)

```
bot = Bot(config_file="my_config.json")

# No configuration file
bot = Bot(
    config_dict={
        "bot": {
            "plugin_dirs": ["plugins"],
            "adapters": ["iamai.adapter.cqhttp"],
        }
    }
)
```

CHAPTER

THREE

PLUGIN BASICS

3.1 Loading Plugins

In the *Getting Started* chapter, we created a `plugins` directory and set it as the plugin directory in the configuration file. Any Python modules placed in the plugins directory (except those starting with _) will be automatically loaded and tested as plugins. Alternatively, you can load plugins "programmatically" without configuring `plugin_dirs` using the following methods.

However, in most cases, you do not need to use the methods below, and it is recommended to load plugins through the configuration options `plugin_dirs` or `plugins`.

3.2 Loading Plugin Directories

Add the following lines in the `main.py` file:

```
from iamai import Bot

bot = Bot()
bot.load_plugins_from_dirs(["plugins", "/home/xxx/iamai/plugins"])

if __name__ == "__main__":
    bot.run()
```

This is actually the same as configuring plugin_dirs to ["plugins", "/home/xxx/iamai/plugins"]. The directories can be relative paths or absolute paths. Plugins starting with _ will not be loaded.

3.3 Loading Individual Plugins

Add the following lines in the `main.py` file:

```
from iamai import Bot

class TestPlugin(Plugin):
   pass
```

(continues on next page)

(continued from previous page)

```
bot = Bot()
bot.load_plugins("plugins.hello", TestPlugin)

if __name__ == "__main__":
    bot.run()
```

The `load_plugins` method can take a plugin class, a string, or a `pathlib.Path` object. If it's the latter two, it will be considered as the name of the plugin module (in the same format as Python's `import` statement) and the path of the plugin module file, respectively.

3.4 Writing Plugins

Each plugin is a plugin class.

Although not mandatory, it is recommended to place each plugin class in a separate Python module.

```
.
├── plugins
│   ├── a.py (single Python file)
│   └── b (Python package)
│       └── __init__.py
├── config.toml
└── main.py
```

Plugin classes must be subclasses of the Plugin class and must implement the `rule()` and `handle()` methods.

```
from iamai import Plugin


class TestPlugin(Plugin):
    priority: int = 0
    block: bool = False

    async def handle(self) -> None:
        pass

    async def rule(self) -> bool:
        return True
```

While the class name `TestPlugin` is not mandatory, it is recommended to use meaningful names for better readability.

The priority attribute represents the priority of the plugin, where a smaller number indicates a higher priority.

The block attribute determines whether to stop event propagation after the current plugin is executed. If set to `True`, event propagation will stop after the current plugin completes, and plugins with lower priorities will not be executed.

Both priority and block are optional, and their default values are 0 and False, respectively.

As mentioned in the section How Does it Work?, when an adapter generates an event (e.g., the robot receives a message), the event will be dispatched to various plugins based on their priorities. iamai will then execute the rule() method of each plugin one by one to determine whether the handle() method should be executed.

The `Plugin` class has built-in attributes and methods:

- `self.event`: The event currently being processed by this plugin.
- `self.name`: The name of the plugin class.
- `self.bot`: The robot object.
- `self.config`: The plugin configuration.
- `self.state`: The plugin state.
- `self.stop()`: Stops the current event propagation.
- `self.skip()`: Skips itself and continues event propagation. All attributes and methods except for `self.event` will be discussed in detail in the section *Advanced Plugins* .

Different adapters generate different events. In the following example, we will write a "Hello" plugin using the CQHTTP adapter.

3.5 Writing the `rule()` Method

```
from iamai import Plugin

class Halloiamai(Plugin):
    async def handle(self) -> None:
        pass

    async def rule(self) -> bool:
        return (
            self.event.adapter.name == "cqhttp"
            and self.event.type == "message"
            and str(self.event.message).lower() == "hello"
        )
```

If you find putting all the conditions in one line less readable, you can write it like this:

```
from iamai import Plugin


class Halloiamai(Plugin):
    async def handle(self) -> None:
        pass

    async def rule(self) -> bool:
        if self.event.adapter.name != "cqhttp":
            return False
        if self.event.type != "message":
            return False
        return str(self.event.message).lower() == "hello"
```

As different adapters generate different events, you should first check the name of the adapter that generated the current event.

Then, check the type of the current event. For CQHTTP adapter, the event types are `message`, `notice`, and `request`, where only message type has the `message` attribute. This plugin only responds to message events.

The `message` attribute of CQHTTP adapter message events represents the received message and is of type `CQHTTPMessage`, which is a subclass of the built-in Message class in iamai.

The built-in Message class in iamai implements many useful methods. It is recommended that all adapter developers use it as much as possible. Specific usage is mentioned in the *Advanced Plugins* section. Here, you can directly use the `str()` function to convert the Message object `self.event.message` to a string.

In addition, commonly used methods in the `rule()` method are `self.event.message.startswith('xxx')` and `self.event.message.endswith('xxx')`, which are equivalent to the `startswith()` and `endswith()` methods of strings.

3.6 Writing the `handle()` method

```
from iamai import Plugin


class Halloiamai(Plugin):
    async def handle(self) -> None:
        await self.event.reply("Hello, iamai!")

    async def rule(self) -> bool:
        if self.event.adapter.name != "cqhttp":
            return False
        if self.event.type != "message":
            return False
        return str(self.event.message).lower() == "hello"
```

As mentioned above, when the rule() method returns `True`, the `handle()` method will be called. Here, we used a method of the message event, `reply()`, to quickly reply to the current message without specifying the recipient of the message.

The `reply()` method is an asynchronous method, so you must use `await` when calling it to wait for it to return.

Now let's look at another example to learn more usage.

3.7 Example : Weather Plugin

```
from iamai import Plugin
from iamai.exceptions import GetEventTimeout


class Weather(Plugin):
    async def handle(self) -> None:
        args = self.event.get_plain_text().split(" ")
        if len(args) >= 2:
            await self.event.reply(await self.get_weather(args[1]))
        else:
```

(continues on next page)

(continued from previous page)

```
            await self.event.reply("Please enter the city you want to query:")
            try:
                city_event = await self.event.adapter.get(
                    lambda x: x.type == "message", timeout=10
                )
            except GetEventTimeout:
                return
            else:
                await self.event.reply(
                    await self.get_weather(city_event.get_plain_text())
                )

    async def rule(self) -> bool:
        if self.event.adapter.name != "cqhttp":
            return False
        if self.event.type != "message":
            return False
        return self.event.message.startswith("weather")

    @staticmethod
    async def get_weather(city):
        if city not in ["Beijing", "Shanghai"]:
            return "The city you want to query is not supported yet!"
        return f"The weather in {city} is..."
```

You can send a message to the robot in the format weather Beijing to get the weather information for Beijing. Alternatively, you can just send weather, and the robot will ask you for the city to query, then you can send the name of the city to query the weather.

In this example, the plugin needs to get the next received message. We use the get() method for this, which is used to obtain events that meet certain conditions. It is also an asynchronous method, so use await to wait for it.

Note that the get() method can specify a timeout to avoid waiting for events indefinitely. When a timeout occurs, it will raise the iamai.exception.GetEventTimeout exception. Be sure to handle this situation properly.

Also, the get_weather() method here does not actually fetch real weather data. You can use any weather API, but when making network requests, use asynchronous libraries like aiohttp or httpx, rather than synchronous ones like requests, to avoid blocking the program.

By reading up to this point, you should be able to write an iamai plugin. Next, we suggest you continue reading Advanced Plugins and the tutorial for the adapter you are going to use.

CHAPTER

FOUR

BUILT-IN MESSAGES

iamai has built-in a message class and recommends all adapter developers to use it as much as possible. It provides many useful functionalities for conveniently constructing rich-text messages.

iamai has built-in the `Message` and `MessageSegment` classes, which represent messages and message segments, respectively.

Most adapter message classes are subclasses of the built-in message classes, but there are some special use cases that can be referred to in the adapter documentation.

The built-in message classes and message segment classes are essentially implementations of the OneBot protocol message classes.

4.1 Message Class

The message class (`Message`) is a subclass of `list` and can be regarded as a list of message segments, but it provides the following additional functionalities:

It overrides the `__init__()` method to allow initialization with objects of types: str, Mapping, Iterable[Mapping], MessageSegment, and Message. Note that str is not natively supported and needs to be implemented by the adapter developer. When a Message object of the same type is passed in during initialization, a new Message object with the same content will be created. When a MessageSegment object is passed in, it will be added to the list. Mapping and Iterable[Mapping] are mainly for the convenience of using pydantic to process events in the adapter, which regular users do not need to be concerned about.

```
msg_seg = MessageSegment()
mas_seg.type = "text"
msg_seg["text"] = "Hello"
msg = Message(msg_seg)

msg = Message("Hello")  # The native built-in Message does not support this usage.

msg = Message(msg)
```

It implements the + and += operators, allowing direct addition with objects of types: `Message`, `MessageSegment`, and `str`.

```
msg = Message()

msg_seg = MessageSegment()
msg_seg.type = "text"
```

(continues on next page)

(continued from previous page)

```
msg_seg["text"] = "Hello"

msg += msg_seg
msg = msg + "Hello" # The native built-in Message does not support this usage.
```

It implements the `startswith()`, `endswith()`, and `replace()` methods, similar to the corresponding methods for strings, but it can accept `MessageSegment` or `str` objects as arguments. Please refer to the API documentation(/api/message.md) for details.

```
msg.startswith("a")
```

4.2 Message Segment

The message segment `class (MessageSegment)` is a data class that also inherits from Mapping. The reason for not using pydantic's model class is to facilitate conversion to JSON in the adapter.

It has two fields: `type` and `data`, which represent the type and content of the message segment, respectively. type is of type `str`, and data is a `dict`. You can directly use dictionary-related operations on the `MessageSegment object`, which is equivalent to operating on the data field. For example:

```
msg_seg = MessageSegment()
msg_seg.type = "text"

msg_seg["text"] = "Hello" # Equivalent to msg_seg.data['text'] = 'Hello'
print(msg_seg.get("text")) # Equivalent to print(msg_seg.data.get('text'))
```

Message segment objects can also be directly added to other objects, and it will return a message class.

```

msg_seg_1 = MessageSegment()
msg_seg_2 = MessageSegment()
msg = msg_seg_1 + msg_seg_2
type(msg)  # Message
```

4.3 Example

```
from iamai import Plugin
from iamai.adapter.cqhttp.message import CQHTTPMessage, CQHTTPMessageSegment

class Hello(Plugin):
    async def handle(self) -> None:
        msg = CQHTTPMessage()
        msg += CQHTTPMessageSegment.text("Hello")
        msg += CQHTTPMessageSegment.image("file//path/hello.png")
        await self.event.reply(msg)

    async def rule(self) -> bool:
        if self.event.adapter.name != "cqhttp":
            return False
```

(continues on next page)

(continued from previous page)

```
        if self.event.type != "message":
            return False
        return str(self.event.message) == "hello"
```

CHAPTER

FIVE

ADVANCED PLUGIN FEATURES

5.1 Event Propogation Control

Sometimes, we may need to control the propagation of events. Besides the basic `block` attribute, which decides whether the event propagation continues after this plugin execution, iamai provides some methods for advanced logic control.

5.1.1 `skip()` Method

The `skip()` method is used to skip the current plugin and continue event propagation. Typically, you could achieve a similar effect using the `return` statement in the `handle()` method. However, the `skip()` method simplifies certain operations in some cases. For example:

```
from iamai import Plugin


class TestPlugin(Plugin):
    async def handle(self) -> None:
        await self.foo()

    async def rule(self) -> bool:
        return True

    async def foo(self):
        self.skip()
```

The `skip()` method can be called from any method within the plugin class, and it takes immediate effect.

5.1.2 `stop()` Method

The `stop()` method is used to end the current event propagation. However, please note that when this method is called, the current event propagation is not immediately terminated. Plugins with the same priority as the current one will still be executed. In other words, the `stop()` method prevents plugins with lower priority from being executed.

5.1.3 `stop()` Method and `block` Attribute

You may notice that setting the block attribute to `True` and adding a `self.stop()` statement at the end of the `handle()` method do not have much difference. In most cases, they are indeed equivalent, except for one scenario. When an exception occurs in the handle() method, the final self.stop() statement will not be executed, but the block attribute will still take effect. In other words, setting block to True and the following example are roughly equivalent:

CHAPTER

SIX

HOT RELOAD

6.1 Manual Cold Reload

iamai provides a `restart()` method to exit and restart the iamai. You can write a plugin like this to restart iamai:

```
from iamai import Plugin

class Restart(Plugin):
    async def handle(self)-> None:
       self.bot.restart()

    async def rule(self) -> bool:
        if self.event.adapter.name != "cqhttp":
           return False

        if self.event.type != "message":
           return False

        return self.event.message.get_plain_text() == "restart"
```

One detail to note is that using this function and manually exiting and restarting iamai have some subtle differences, mainly in that using this function will not clear plugin and global states.

6.2 Reload Plugin

In addition, iamai also provides a `reload_plugins()` method to reload all plugins. This method does not reload configuration files, adapters, etc.

6.3 Automatic Hot Reload

Since version 0.4.0, iamai has supported automatic hot reload. This means that when plugin files or configuration files are updated, iamai does not need to be restarted, and the changes will be automatically loaded.

This feature requires support from the `watchfiles` library, so please manually install this Python library.

When the configuration file is updated, the configuration file will be reloaded. If the bot table changes, the `restart()` method will be called to restart iamai.

When plugin files in the directories set in `plugin_dirs` are added, modified, or deleted, iamai will automatically try to import, reload, or delete the corresponding plugins.

Enabling this feature is very simple, just pass the `hot_reload` parameter when instantiating the `Bot` class.

```
from iamai import Bot

bot = Bot(hot_reload=True)

if __name__ == "__main__":
    bot.run()
```

However, please note that this feature is still in early experimental stage. If you encounter any issues during use, please provide feedback. Also, this feature may slightly affect performance, so if your use case is highly performance-sensitive, do not enable this feature in production environments.

CHAPTER

SEVEN

SCHEDULED TASKS

iamai's scheduled tasks are implemented using the APScheduler library. iamai wraps it into an adapter, providing the same interface as other regular adapters.

7.1 Downloading and Loading the Adapter

Like other adapters, the scheduled task adapter also needs to be downloaded separately:

```
pip install iamai-adapter-apscheduler
```

Alternatively, you can install it along with iamai:

```
pip install iamai[apscheduler]
```

Once downloaded, you can configure it in the `config.toml` file just like any other adapter:

```
[bot]
adapters = ["iamai.adapter.xxxx", "iamai.adapter.apscheduler"]
```

Or you can manually load it:

```

from iamai import Bot

bot = Bot()
bot.load_adapters("iamai.adapter.apscheduler")

if __name__ == "__main__":
    bot.run()

```

However, please note that when manually loading the adapter, you need to load the adapter after loading all the plugins.

7.2 Configuring APScheduler

The `iamai-adapter-apscheduler` adapter has only one configuration item, `scheduler_config`:

```
[bot]
adapters = ["iamai.adapter.xxxx", "iamai.adapter.apscheduler"]

[adapter.apscheduler]
scheduler_config = { "apscheduler.timezone" = "Asia/Shanghai" }
```

Please refer to the APScheduler documentation for specific configuration options: scheduler-config .

7.3 Usage

7.3.1 Adding Attributes Directly

After loading the adapter, for plugins that need to be executed on a schedule, they should have a format similar to the following:

```
from iamai import Plugin


class TestPlugin(Plugin):
    __schedule__ = True
    trigger = "interval"
    trigger_args = {"seconds": 10}

    async def handle(self) -> None:
        pass

    async def rule(self) -> bool:
        return (
            self.event.adapter.name == "apscheduler"
            and type(self) == self.event.plugin_class
        )
```

Scheduled plugins must have the `__schedule__`, `trigger`, and `trigger_args` attributes set.

`__schedule__` must be set to True. `trigger` represents the APScheduler trigger. `trigger_args` represents the adapter configuration. They will be passed to the `add_job()` method of the scheduler in the following form:

```
scheduler.add_job(func, trigger, **trigger_args)
```

The APScheduler adapter will iterate over all plugins during startup, looking for plugins that meet the above criteria, and add scheduled tasks based on the plugin descriptions.

When a scheduled task is triggered, the APScheduler adapter will generate an event and perform normal event propagation.

This event's `type` attribute is always apscheduler, and it has a `plugin_class` attribute representing the plugin class that defined this scheduled task. You can use `type(self) == self.event.plugin_class` in the `rule()` method to ensure that this plugin only handles its own defined scheduled events.

7.4 Using Class Decorators (Recommended)

In addition to the above method, you can also use class decorators to decorate an existing plugin as a scheduled task plugin.

```
from iamai import Plugin
from iamai.adapter.apscheduler import scheduler_decorator


@scheduler_decorator(
    trigger="interval", trigger_args={"seconds": 10}, override_rule=True
)
class TestPlugin(Plugin):
    async def handle(self) -> None:
        pass

    async def rule(self) -> bool:
        if self.event.adapter.name != "cqhttp":
            return False
        if self.event.type != "message":
            return False
        return str(self.event.message).lower() == "hello"
```

The above syntax is equivalent to:

```
from iamai import Plugin


class TestPlugin(Plugin):
    __schedule__ = True
    trigger = "interval"
    trigger_args = {"seconds": 10}

    async def handle(self) -> None:
        pass

    async def rule(self) -> bool:
        if self.event.name == "apscheduler" and type(self) == self.event.plugin_class:
            return True
        else:
            if self.event.adapter.name != "cqhttp":
                return False
            if self.event.type != "message":
                return False
            return str(self.event.message).lower() == "hello"
```

As shown in the above example, setting the `override_rule` parameter to True can automatically override the `rule()` method and add the logic to handle the scheduled task events.

7.5 Tips

When the same scheduled task needs to wake up multiple plugins simultaneously, and multiple scheduled tasks are set, and you do not want them to interfere with each other, you can do as follows:

First Plugin:

```
from iamai import Plugin
from iamai.adapter.apscheduler import scheduler_decorator


@scheduler_decorator(
    trigger="interval", trigger_args={"seconds": 10}, override_rule=False
)
class PluginA(Plugin):
    scheduler_flag = "abc"

    async def handle(self) -> None:
        pass

    async def rule(self) -> bool:
        if (
            self.event.type == "apscheduler"
            and getattr(self.event.plugin_class, "scheduler_flag", "") == "abc"
        ):
            return True
        else:
            pass
```

Second Plugin:

```
from iamai import Plugin


class PluginB(Plugin):
    async def handle(self) -> None:
        pass

    async def rule(self) -> bool:
        if (
            self.event.type == "apscheduler"
            and getattr(self.event.plugin_class, "scheduler_flag", "") == "abc"
        ):
            return True
        else:
            pass
```

You need to set a unique `scheduler_flag` attribute for the plugin that defines the scheduled task. Of course, this attribute can also be named anything else. Then the second plugin can determine the value of `scheduler_flag` to judge whether the current event is a scheduled event defined by the first plugin.

7.6 Example

Now let's try using the CQHTTP protocol adapter to send a message on a schedule.

```
from time import strftime, localtime

from iamai import Plugin
from iamai.adapter.apscheduler import scheduler_decorator


@scheduler_decorator(
    trigger="interval", trigger_args={"seconds": 100}, override_rule=True
)
class Schedule(Plugin):
    async def handle(self) -> None:
        await self.bot.get_adapter("cqhttp").send(
            f"Time: {strftime('%Y-%m-%d %H:%M:%S', localtime())}",
            message_type="group",
            id_=1234567890,  # Group ID
        )

    async def rule(self) -> bool:
        return False
```

CHAPTER

EIGHT

HOOK FUNCTIONS

iamai provides same hook functions that can be used as decorators. Here's how to use them:

```
from iamai import Bot

bot = Bot()

@bot.bot_run_hook
async def hook_func(_bot: Bot):
    pass

if __name__ == "__main__":
    bot.run()
```

::: tip Note If you are not sure what you are doing, please do not add hook functions. :::

8.1 Bot-Related Hooks

8.1.1 Bot Startup

```
@bot.bot_run_hook
async def hook_func(_bot: Bot):
    pass
```

8.1.2 Bot Exit

```
@bot.bot_exit_hook
async def hook_func(_bot: Bot):
    pass
```

::: warning Note This hook function is not a coroutine! :::

8.2 Adapter-Related Hooks

8.2.1 Adapter Initialization

```
@bot.adapter_startup_hook
async def hook_func(_adapter: "T_Adapter"):
  pass
```

8.2.2 Adapter Run

```
@bot.adapter_run_hook
async def hook_func(_adapter: "T_Adapter"):
    pass
```

8.2.3 Adapter Shutdown

```
@bot.adapter_shutdown_hook

async def hook_func(_adapter : "T_Adapter"):
   pass
```

8.3 Event Processing-Related Hooks

8.3.1 Event Processing

```
@bot.event_preprocessor_hook
async def hook_func(_event: "T_Event"):
   pass
```

8.4 Event PostProcessing

```
@bot.event_postprocessor_hook
async def hook_func(_event: "T_Event"):
   pass
```

CHAPTER

NINE

CQHTTP PROTOCOL ADAPTER

9.1 Installation

```
pip install iamai-adapter-cqhttp
```

9.2 Configuring the Protocol Endpoint

The CQHTTP protocol adapter is an adapter for the OneBot protocol (formerly known as the CKYU platform's CQHTTP protocol). It requires a protocol endpoint that is compatible with the OneBot protocol for communication. Here are some commonly used QQ protocol endpoints that support the OneBot protocol:

- go-cqhttp
- mirai + onebot-kotlin
- oicq

Below is an example using go-cqhttp:

1. Download the release version of go-cqhttp corresponding to your platform: Github Releases
2. Run `go-cqhttp` or `go-cqhttp.exe` to generate the default configuration file.
3. Edit the configuration file `config.yml` and restart the program.

Currently, the CQHTTP adapter supports WebSocket connection and reverse WebSocket connection. The go-cqhttp configuration file should look like this:

```
account:
   uin: BotQQNumber
   password:'BotPassword'

message:
   # Post data format
   post-format: array

# Connection service list, choose only one of the following reverse WS and forward WS

servers:
  # Reverse WS settings
  - ws-reverse:
      # Reverse WS Universal address
```

(continues on next page)

(continued from previous page)

```
    universal: ws://127.0.0.1:8080/cqhttp/ws
    # Reconnection interval in milliseconds
    reconnect-interval: 3000
    middlewares:
      <<: *default # Reference default middleware
# Forward WS settings
- ws:
    # Forward WS server listening address
    host: 127.0.0.1
    # Forward WS server listening port
    port: 6700
    middlewares:
      <<: *default # Reference default middleware
```

Other items can remain as default.

9.3 Configuring iamai

If you have installed and configured `go-cqhttp` as mentioned above and are using reverse WebSocket connection, there is no need to configure iamai separately.

If you have other specific requirements, you can edit `config.toml` for configuration, refer to *Basic Configuration* and *CQHTTP Configuration*

9.4 Running Tests

```
2021-09-01 18:05:29.740 | INFO     | iamai.adapter.cqhttp:handle_cqhttp_event:138 -␣
↪WebSocket connection from CQHTTP Bot xxxxxx accepted!
```

9.5 Sending Rich Text Messages

When writing plugins, besides sending regular text messages, you can also easily construct and send rich-text messages. Make sure you have read the Built-in Messages section before reading this section.

```
from iamai import Plugin
from iamai.adapter.cqhttp.message import CQHTTPMessageSegment

class Helloiamai(Plugin):
    async def handle(self) -> None:
        msg = CQHTTPMessageSegment.text("Hello, iamai!") + \
              CQHTTPMessageSegment.image("https://www.example.org/1.jpg")
        await self.event.reply(msg)

    async def rule(self) -> bool:
        if self.event.adapter.name != "cqhttp":
            return False
        if self.event.type != "message":
```

(continues on next page)

(continued from previous page)

```
            return False
        return str(self.event.message).lower() == "hello"
```

For more usage methods, please refer to OneBot Message Segment Types and *CQHTTP Messages* .

9.6 Calling OneBot API

You can call OneBot API using following method :

```
from iamai import Plugin

class TestPlugin(Plugin):
    async def handle(self) -> None:
        resp = await self.event.adapter.call_api("send_like", user_id=10001)
        # Equivalent to resp = await self.event.adapter.send_like(user_id=10001)

    async def rule(self) -> bool:
        return self.event.adapter.name == "cqhttp"
```

For more usage methods, please refer to OneBot Public API .

CHAPTER

TEN

MIRAI PROTOCOL ADAPTER

The Mirai Protocol Adapter is an adapter for the mirai-api-http protocol. You need to install mirai-api-http following the instructions in its documentation.

Please note that this adapter only supports mirai-api-http version 2.3 and above.

This adapter supports both the Websocket Adapter mode and the Reverse Websocket Adapter mode of mirai-api-http.

10.1 Installation

```
pip install iamai-adapter-mirai
```

10.2 Configuring the Protocol Endpoint

Edit the `setting.yml` configuration file of mirai-api-http:

10.2.1 Websocket Adapter Mode

```yaml
adapters:
  - ws
enableVerify: true
verifyKey: 1234567890
adapterSettings:
  ws:
    host: localhost
    port: 8080
    reservedSyncId: -1
```

10.2.2 Reverse Websocket Adapter Mode

```yaml
adapters:
  - reverse-ws
enableVerify: true
verifyKey: 1234567890
singleMode: true
adapterSettings:
  reverse-ws:
    destinations:
    - host: localhost
      port: 8080
      path: /mirai/ws
      protocol: ws
      method: GET
    reservedSyncId: -1
```

10.3 Configuring iamai

Edit the `config.toml` configuration file of iamai.

For more options, please refer to the Mirai configuration.

10.3.1 Websocket Adapter Mode

```toml
[bot]
adapters = ["iamai.adapter.mirai"]

[adapter.mirai]
adapter_type = "ws"
verify_key = "1234567890"
qq = "QQ"
```

10.3.2 Reverse Websocket Adapter Mode

```toml
[bot]
adapters = ["iamai.adapter.mirai"]

[adapter.mirai]
adapter_type = "reverse-ws"
verify_key = "1234567890"
qq = "QQ"
```

10.4 Sending Rich Text Messages

Similar to the CQHTTP protocol adapter, the Mirai adapter can easily construct rich text messages.

```
from iamai import Plugin
from iamai.adapter.mirai.message import MiraiMessageSegment


class Halloiamai(Plugin):
    async def handle(self) -> None:
        msg = MiraiMessageSegment.plain("Hello, iamai!") + \
              MiraiMessageSegment.image(url="https://www.example.org/1.jpg")
        await self.event.reply(msg)

    async def rule(self) -> bool:
        if self.event.adapter.name != "mirai":
            return False
        if self.event.type != "FriendMessage":
            return False
        return self.event.message.get_plain_text().lower() == "hello"
```

For more usage, please refer to mirai-api-http message segment types and Mirai messages.

10.5 Calling mirai-api-http API

You can use the following method to call mirai-api-http API:

```
from iamai import Plugin


class TestPlugin(Plugin):
    async def handle(self) -> None:
        resp = await self.event.adapter.call_api("friendProfile", target=10001)
        # Equivalent to resp = await self.event.adapter.friendProfile(target=10001)

    async def rule(self) -> bool:
        return self.event.adapter.name == "mirai"
```

For more usage, please refer to mirai-api-http documentation .

CHAPTER

ELEVEN

DINGTALK PROTOCOL ADAPTER

11.1 Installation

```
pip install iamai-adapter-dingtalk
```

11.2 Configuring the Protocol Endpoint

The DingTalk protocol adapter is an adapter for the DingTalk enterprise robot protocol. DingTalk's enterprise robot uses the outgoing (callback) mechanism. After a user mentions the robot, DingTalk will POST the message content to the developer's message receiving address.

For specific configurations, refer to the relevant documentation on the DingTalk Open Platform:

- Robot Overiew
- Enterprise Internal Development of Robots

During testing, you may not have your own public domain name or IP. DingTalk provides an intranet penetration tool: Intranet Penetration Tool .

11.3 Configuring iamai

You need to edit `config.toml` to configure the DingTalk adapter. Refer to *Basic Configuration* and *DingTalk Configuration* .

11.4 Sending Rich Text Messages

When writing plugins, besides sending regular text messages, you can also easily construct and send rich-text messages. Make sure you have read the Built-in Messages section before reading this section.

The special thing is that due to the uniqueness of DingTalk's rich-text messages, the message class of the DingTalk adapter `DingTalkMessage` is not a subclass of `Message`, but a subclass of `MessageSegment`. You cannot build messages by simply adding common message fields. Instead, you need to manually write Markdown text.

```
from iamai import Plugin
from iamai.adapter.digntalk.message import DingTalkMessage
```

(continues on next page)

(continued from previous page)

```
class Halloiamai(Plugin):
    async def handle(self) -> None:
        await self.event.reply(DingTalkMessage.markdown("# hello\n\nHello, iamai!"))

    async def rule(self) -> bool:
        if self.event.adapter.name != "dingtalk":
            return False
        return str(self.event.message).strip().lower() == "hello"
```

For more usage methods, please refer to Message Types and Data Formats and *CQHTTP Messages* .

CHAPTER

TWELVE

IAMAI

12.1 iamai package

12.1.1 Subpackages

iamai.adapter package

Subpackages

iamai.adapter.apscheduler package

Submodules

iamai.adapter.apscheduler.config module

iamai.adapter.apscheduler.event module

Module contents

iamai.adapter.bililive package

Submodules

iamai.adapter.bililive.config module

iamai.adapter.bililive.event module

iamai.adapter.bililive.exceptions module

iamai.adapter.bililive.message module

iamai.adapter.bililive.tests module

Module contents

iamai.adapter.console package

Submodules

iamai.adapter.console.config module

Console

class iamai.adapter.console.config.**Config**(*, *show_raw: bool = False*, ***extra_data: Any*)

Bases: *ConfigModel*

Console

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'show_raw': FieldInfo(annotation=bool, required=False, default=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

show_raw: bool

iamai.adapter.console.event module

iamai.adapter.console.message module

class iamai.adapter.console.message.**ConsoleMessage**(*, *type: str*, *data: Dict[str, Any] = None*)

Bases: `MessageSegment[NoneType]`

Console

is_text() → bool

Returns

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'data': FieldInfo(annotation=Dict[str, Any], required=False, default_factory=dict), 'type': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

iamai.adapter.console.message.**escape_tag**(*s: str*) → str

<*tag*>

: [loguru color](https://loguru.readthedocs.io/en/stable/api/logger.html#color)

:

s:

Module contents

iamai.adapter.cqhttp package

Submodules

iamai.adapter.cqhttp.config module

iamai.adapter.cqhttp.event module

iamai.adapter.cqhttp.exceptions module

iamai.adapter.cqhttp.message module

Module contents

iamai.adapter.gensokyo package

Submodules

iamai.adapter.gensokyo.config module

GSK

class iamai.adapter.gensokyo.config.**Config**(*, *adapter_type: Literal['ws', 'reverse-ws', 'ws-reverse'] = 'reverse-ws'*, *host: str = '127.0.0.1'*, *port: int = 8080*, *url: str = '/gsk/ws'*, *reconnect_interval: int = 3*, *api_timeout: int = 1000*, *app_id: str = ''*, *app_secret: str = ''*, *token: str = ''*, *access_token: str = ''*, ***extra_data: Any*)

Bases: *ConfigModel*

GSK

adapter_type

Type
Literal[‘ws’, ‘reverse-ws’, ‘ws-reverse’]

host

Type
str

port

Type
int

url
WebSocket

Type
str

reconnect_interval

Type
int

api_timeout
API

Type
int

access_token

Type
str

access_token: str

adapter_type: Literal['ws', 'reverse-ws', 'ws-reverse']

api_timeout: int

app_id: str

app_secret: str

host: str

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}
Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'access_token': FieldInfo(annotation=str, required=False, default=''), 'adapter_type': FieldInfo(annotation=Literal['ws', 'reverse-ws', 'ws-reverse'], required=False, default='reverse-ws'), 'api_timeout': FieldInfo(annotation=int, required=False, default=1000), 'app_id': FieldInfo(annotation=str, required=False, default=''), 'app_secret': FieldInfo(annotation=str, required=False, default=''), 'host': FieldInfo(annotation=str, required=False, default='127.0.0.1'), 'port': FieldInfo(annotation=int, required=False, default=8080), 'reconnect_interval': FieldInfo(annotation=int, required=False, default=3), 'token': FieldInfo(annotation=str, required=False, default=''), 'url': FieldInfo(annotation=str, required=False, default='/gsk/ws')}
Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

port: int

reconnect_interval: int

token: str

url: str

iamai.adapter.gensokyo.event module

GSK

class iamai.adapter.gensokyo.event.**Anonymous**(*, *id: int*, *name: str*, *flag: str*)

Bases: BaseModel

flag: str

id: int

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'flag': FieldInfo(annotation=str, required=True), 'id': FieldInfo(annotation=int, required=True), 'name': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

name: str

class iamai.adapter.gensokyo.event.**File**(*, *id: str*, *name: str*, *size: int*, *busid: int*)

Bases: BaseModel

busid: int

id: str

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'busid': FieldInfo(annotation=int, required=True), 'id': FieldInfo(annotation=str, required=True), 'name': FieldInfo(annotation=str, required=True), 'size': FieldInfo(annotation=int, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

name: str

size: int

class iamai.adapter.gensokyo.event.**FriendAddNoticeEvent**(*, *adapter: Any*, *post_type: str | None*, *time: int*, *self_id: int*, *notice_type: Literal['friend_add']*, *user_id: int*, ***extra_data: Any*)

Bases: *NoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'notice_type': FieldInfo(annotation=Literal['friend_add'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=int, required=True), 'time': FieldInfo(annotation=int, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True, alias='post_type', alias_priority=2), 'user_id': FieldInfo(annotation=int, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['friend_add']

user_id: int

class iamai.adapter.gensokyo.event.**FriendRecallNoticeEvent**(*, *adapter: Any*, *post_type: str | None*, *time: int*, *self_id: int*, *notice_type: Literal['friend_recall']*, *user_id: int*, *message_id: int*, ***extra_data: Any*)

Bases: *NoticeEvent*

message_id: int

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'message_id': FieldInfo(annotation=int, required=True), 'notice_type': FieldInfo(annotation=Literal['friend_recall'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=int, required=True), 'time': FieldInfo(annotation=int, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True, alias='post_type', alias_priority=2), 'user_id': FieldInfo(annotation=int, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['friend_recall']

user_id: int

class iamai.adapter.gensokyo.event.**FriendRequestEvent**(*, *adapter: Any*, *post_type: str | None*, *time: int*, *self_id: int*, *request_type: Literal['friend']*, *user_id: int*, *comment: str*, *flag: str*, ***extra_data: Any*)

Bases: *RequestEvent*

async approve(*remark: str = ''*) → Dict[str, Any]

Parameters
remark –

Returns
API

comment: str

flag: str

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'comment': FieldInfo(annotation=str, required=True), 'flag': FieldInfo(annotation=str, required=True), 'post_type': FieldInfo(annotation=Literal['request'], required=True), 'request_type': FieldInfo(annotation=Literal['friend'], required=True), 'self_id': FieldInfo(annotation=int, required=True), 'time': FieldInfo(annotation=int, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True, alias='post_type', alias_priority=2), 'user_id': FieldInfo(annotation=int, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

async refuse() → Dict[str, Any]

Returns
API

request_type: Literal['friend']

user_id: int

class iamai.adapter.gensokyo.event.GSKEvent(*, *adapter: Any*, *post_type: str | None*, *time: int*, *self_id: int*, ***extra_data: Any*)

Bases: Event[GSKAdapter]

GSK

classmethod get_event_type() → Tuple[str | None, str | None, str | None]

Returns

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}
Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'post_type': FieldInfo(annotation=str, required=True), 'self_id': FieldInfo(annotation=int, required=True), 'time': FieldInfo(annotation=int, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True, alias='post_type', alias_priority=2)}
Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

post_type: str

self_id: int

time: int

property to_me: bool
user_id *self_id*

type: str | None

class iamai.adapter.gensokyo.event.GroupAdminNoticeEvent(*, *adapter: Any*, *post_type: str | None*, *time: int*, *self_id: int*, *notice_type: Literal['group_admin']*, *sub_type: Literal['set', 'unset']*, *user_id: int*, *group_id: int*, ***extra_data: Any*)

Bases: *NoticeEvent*

group_id: int

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'notice_type': FieldInfo(annotation=Literal['group_admin'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=int, required=True), 'sub_type': FieldInfo(annotation=Literal['set', 'unset'], required=True), 'time': FieldInfo(annotation=int, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True, alias='post_type', alias_priority=2), 'user_id': FieldInfo(annotation=int, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['group_admin']

sub_type: Literal['set', 'unset']

user_id: int

class iamai.adapter.gensokyo.event.**GroupBanNoticeEvent**(*, *adapter: Any*, *post_type: str | None*, *time: int*, *self_id: int*, *notice_type: Literal['group_ban']*, *sub_type: Literal['ban', 'lift_ban']*, *group_id: int*, *operator_id: int*, *user_id: int*, *duration: int*, ***extra_data: Any*)

Bases: *NoticeEvent*

duration: int

group_id: int

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'duration': FieldInfo(annotation=int, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'notice_type': FieldInfo(annotation=Literal['group_ban'], required=True), 'operator_id': FieldInfo(annotation=int, required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=int, required=True), 'sub_type': FieldInfo(annotation=Literal['ban', 'lift_ban'], required=True), 'time': FieldInfo(annotation=int, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True, alias='post_type', alias_priority=2), 'user_id': FieldInfo(annotation=int, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['group_ban']

operator_id: int

sub_type: Literal['ban', 'lift_ban']

user_id: int

class iamai.adapter.gensokyo.event.**GroupDecreaseNoticeEvent**(*, *adapter: Any*, *post_type: str | None*, *time: int*, *self_id: int*, *notice_type: Literal['group_decrease']*, *sub_type: Literal['leave', 'kick', 'kick_me']*, *group_id: int*, *operator_id: int*, *user_id: int*, ***extra_data: Any*)

Bases: *NoticeEvent*

group_id: int

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'notice_type': FieldInfo(annotation=Literal['group_decrease'], required=True), 'operator_id': FieldInfo(annotation=int, required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=int, required=True), 'sub_type': FieldInfo(annotation=Literal['leave', 'kick', 'kick_me'], required=True), 'time': FieldInfo(annotation=int, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True, alias='post_type', alias_priority=2), 'user_id': FieldInfo(annotation=int, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['group_decrease']

operator_id: int

sub_type: Literal['leave', 'kick', 'kick_me']

user_id: int

class iamai.adapter.gensokyo.event.**GroupHonorNotifyEvent**(*, *adapter: Any*, *post_type: str | None*, *time: int*, *self_id: int*, *notice_type: Literal['notify']*, *sub_type: Literal['honor']*, *group_id: int*, *user_id: int*, *honor_type: Literal['talkative', 'performer', 'emotion']*, ***extra_data: Any*)

Bases: *NotifyEvent*

group_id: int

honor_type: Literal['talkative', 'performer', 'emotion']

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'honor_type': FieldInfo(annotation=Literal['talkative', 'performer', 'emotion'], required=True), 'notice_type': FieldInfo(annotation=Literal['notify'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=int, required=True), 'sub_type': FieldInfo(annotation=Literal['honor'], required=True), 'time': FieldInfo(annotation=int, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True, alias='post_type', alias_priority=2), 'user_id': FieldInfo(annotation=int, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

sub_type: Literal['honor']

class iamai.adapter.gensokyo.event.**GroupIncreaseNoticeEvent**(*, *adapter: Any*, *post_type: str | None*, *time: int*, *self_id: int*, *notice_type: Literal['group_increase']*, *sub_type: Literal['approve', 'invite']*, *group_id: int*, *operator_id: int*, *user_id: int*, ***extra_data: Any*)

Bases: *NoticeEvent*

group_id: int

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'notice_type': FieldInfo(annotation=Literal['group_increase'], required=True), 'operator_id': FieldInfo(annotation=int, required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=int, required=True), 'sub_type': FieldInfo(annotation=Literal['approve', 'invite'], required=True), 'time': FieldInfo(annotation=int, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True, alias='post_type', alias_priority=2), 'user_id': FieldInfo(annotation=int, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['group_increase']

operator_id: int

sub_type: Literal['approve', 'invite']

user_id: int

class iamai.adapter.gensokyo.event.**GroupLuckyKingNotifyEvent**(*, *adapter: Any*, *post_type: str | None*, *time: int*, *self_id: int*, *notice_type: Literal['notify']*, *sub_type: Literal['lucky_king']*, *group_id: int*, *user_id: int*, *target_id: int*, ***extra_data: Any*)

Bases: *NotifyEvent*

group_id: int

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'notice_type': FieldInfo(annotation=Literal['notify'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=int, required=True), 'sub_type': FieldInfo(annotation=Literal['lucky_king'], required=True), 'target_id': FieldInfo(annotation=int, required=True), 'time': FieldInfo(annotation=int, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True, alias='post_type', alias_priority=2), 'user_id': FieldInfo(annotation=int, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

sub_type: Literal['lucky_king']

target_id: int

class iamai.adapter.gensokyo.event.**GroupMessageEvent**(*, *adapter: Any*, *post_type: str | None*, *time: int*, *self_id: int*, *message_type: Literal['group']*, *sub_type: Literal['normal', 'anonymous', 'notice']*, *message_id: int*, *user_id: int*, *message:* GSKMessage, *raw_message: str*, *font: int*, *sender:* Sender, *group_id: int*, *anonymous:* Anonymous | *None* = *None*, ***extra_data: Any*)

Bases: *MessageEvent*

anonymous: *Anonymous* | None

group_id: int

message_type: Literal['group']

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'anonymous': FieldInfo(annotation=Union[Anonymous, NoneType], required=False), 'font': FieldInfo(annotation=int, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'message': FieldInfo(annotation=GSKMessage, required=True), 'message_id': FieldInfo(annotation=int, required=True), 'message_type': FieldInfo(annotation=Literal['group'], required=True), 'post_type': FieldInfo(annotation=Literal['message'], required=True), 'raw_message': FieldInfo(annotation=str, required=True), 'self_id': FieldInfo(annotation=int, required=True), 'sender': FieldInfo(annotation=Sender, required=True), 'sub_type': FieldInfo(annotation=Literal['normal', 'anonymous', 'notice'], required=True), 'time': FieldInfo(annotation=int, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True, alias='post_type', alias_priority=2), 'user_id': FieldInfo(annotation=int, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

async reply(*message: List[*GSKMessageSegment*]* | GSKMessageSegment | *str* | *Mapping[str, Any]*) → Dict[str, Any]

Parameters
message – *call_api()*

Returns
API

sub_type: Literal['normal', 'anonymous', 'notice']

class iamai.adapter.gensokyo.event.**GroupRecallNoticeEvent**(*, *adapter: Any*, *post_type: str | None*, *time: int*, *self_id: int*, *notice_type: Literal['group_recall']*, *group_id: int*, *operator_id: int*, *user_id: int*, *message_id: int*, ***extra_data: Any*)

Bases: *NoticeEvent*

group_id: int

message_id: int

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'message_id': FieldInfo(annotation=int, required=True), 'notice_type': FieldInfo(annotation=Literal['group_recall'], required=True), 'operator_id': FieldInfo(annotation=int, required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=int, required=True), 'time': FieldInfo(annotation=int, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True, alias='post_type', alias_priority=2), 'user_id': FieldInfo(annotation=int, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['group_recall']

operator_id: int

user_id: int

class iamai.adapter.gensokyo.event.**GroupRequestEvent**(*, *adapter: Any*, *post_type: str | None*, *time: int*, *self_id: int*, *request_type: Literal['group']*, *sub_type: Literal['add', 'invite']*, *group_id: int*, *user_id: int*, *comment: str*, *flag: str*, ***extra_data: Any*)

Bases: *RequestEvent*

/

async approve() → Dict[str, Any]

Returns
API

comment: str

flag: str

group_id: int

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'comment': FieldInfo(annotation=str, required=True), 'flag': FieldInfo(annotation=str, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'post_type': FieldInfo(annotation=Literal['request'], required=True), 'request_type': FieldInfo(annotation=Literal['group'], required=True), 'self_id': FieldInfo(annotation=int, required=True), 'sub_type': FieldInfo(annotation=Literal['add', 'invite'], required=True), 'time': FieldInfo(annotation=int, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True, alias='post_type', alias_priority=2), 'user_id': FieldInfo(annotation=int, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

async refuse(*reason: str = ''*) → Dict[str, Any]

Parameters
reason –

Returns
API

request_type: Literal['group']

sub_type: Literal['add', 'invite']

user_id: int

class iamai.adapter.gensokyo.event.**GroupUploadNoticeEvent**(**, adapter: Any, post_type: str | None, time: int, self_id: int, notice_type: Literal['group_upload'], user_id: int, group_id: int, file:* File, ***extra_data: Any*)

Bases: *NoticeEvent*

file: *File*

group_id: int

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'file': FieldInfo(annotation=File, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'notice_type': FieldInfo(annotation=Literal['group_upload'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=int, required=True), 'time': FieldInfo(annotation=int, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True, alias='post_type', alias_priority=2), 'user_id': FieldInfo(annotation=int, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['group_upload']

user_id: int

class iamai.adapter.gensokyo.event.**HeartbeatMetaEvent**(*, *adapter: Any*, *post_type: str | None*, *time: int*, *self_id: int*, *meta_event_type: Literal['heartbeat']*, *status:* Status, *interval: int*, ***extra_data: Any*)

Bases: *MetaEvent*

interval: int

meta_event_type: Literal['heartbeat']

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'interval': FieldInfo(annotation=int, required=True), 'meta_event_type': FieldInfo(annotation=Literal['heartbeat'], required=True), 'post_type': FieldInfo(annotation=Literal['meta_event'], required=True), 'self_id': FieldInfo(annotation=int, required=True), 'status': FieldInfo(annotation=Status, required=True), 'time': FieldInfo(annotation=int, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True, alias='post_type', alias_priority=2)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

status: *Status*

class iamai.adapter.gensokyo.event.**LifecycleMetaEvent**(*, *adapter: Any*, *post_type: str | None*, *time: int*, *self_id: int*, *meta_event_type: Literal['lifecycle']*, *sub_type: Literal['enable', 'disable', 'connect']*, ***extra_data: Any*)

Bases: *MetaEvent*

meta_event_type: Literal['lifecycle']

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'meta_event_type': FieldInfo(annotation=Literal['lifecycle'], required=True), 'post_type': FieldInfo(annotation=Literal['meta_event'], required=True), 'self_id': FieldInfo(annotation=int, required=True), 'sub_type': FieldInfo(annotation=Literal['enable', 'disable', 'connect'], required=True), 'time': FieldInfo(annotation=int, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True, alias='post_type', alias_priority=2)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

sub_type: Literal['enable', 'disable', 'connect']

class iamai.adapter.gensokyo.event.**MessageEvent**(*, *adapter: Any*, *post_type: str | None*, *time: int*, *self_id: int*, *message_type: Literal['private', 'group']*, *sub_type: str*, *message_id: int*, *user_id: int*, *message:* GSKMessage, *raw_message: str*, *font: int*, *sender:* Sender, ***extra_data: Any*)

Bases: *GSKEvent*, MessageEvent[GSKAdapter]

font: int

get_plain_text() → str

Returns

async is_same_sender(*other: Self*) → bool

Parameters
other –

Returns

message: *GSKMessage*

message_id: int

message_type: Literal['private', 'group']

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'font': FieldInfo(annotation=int, required=True), 'message': FieldInfo(annotation=GSKMessage, required=True), 'message_id': FieldInfo(annotation=int, required=True), 'message_type': FieldInfo(annotation=Literal['private', 'group'], required=True), 'post_type': FieldInfo(annotation=Literal['message'], required=True), 'raw_message': FieldInfo(annotation=str, required=True), 'self_id': FieldInfo(annotation=int, required=True), 'sender': FieldInfo(annotation=Sender, required=True), 'sub_type': FieldInfo(annotation=str, required=True), 'time': FieldInfo(annotation=int, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True, alias='post_type', alias_priority=2), 'user_id': FieldInfo(annotation=int, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

post_type: Literal['message']

raw_message: str

async reply(*message: List[*GSKMessageSegment*] | *GSKMessageSegment | *str | Mapping[str, Any]*) → Dict[str, Any]

Parameters
message – *call_api()*

Returns
API

sender: *Sender*

sub_type: str

user_id: int

class iamai.adapter.gensokyo.event.**MetaEvent**(*, *adapter: Any*, *post_type: str | None*, *time: int*, *self_id: int*, *meta_event_type: str*, ***extra_data: Any*)

Bases: *GSKEvent*

meta_event_type: str

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'meta_event_type': FieldInfo(annotation=str, required=True), 'post_type': FieldInfo(annotation=Literal['meta_event'], required=True), 'self_id': FieldInfo(annotation=int, required=True), 'time': FieldInfo(annotation=int, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True, alias='post_type', alias_priority=2)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

post_type: Literal['meta_event']

class iamai.adapter.gensokyo.event.**NoticeEvent**(*, *adapter: Any*, *post_type: str | None*, *time: int*, *self_id: int*, *notice_type: str*, ***extra_data: Any*)

Bases: *GSKEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'notice_type': FieldInfo(annotation=str, required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=int, required=True), 'time': FieldInfo(annotation=int, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True, alias='post_type', alias_priority=2)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: str

post_type: Literal['notice']

class iamai.adapter.gensokyo.event.**NotifyEvent**(*, *adapter: Any*, *post_type: str | None*, *time: int*, *self_id: int*, *notice_type: Literal['notify']*, *sub_type: str*, *group_id: int | None = None*, *user_id: int*, ***extra_data: Any*)

Bases: *NoticeEvent*

group_id: int | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'group_id': FieldInfo(annotation=Union[int, NoneType], required=False), 'notice_type': FieldInfo(annotation=Literal['notify'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=int, required=True), 'sub_type': FieldInfo(annotation=str, required=True), 'time': FieldInfo(annotation=int, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True, alias='post_type', alias_priority=2), 'user_id': FieldInfo(annotation=int, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['notify']

sub_type: str

user_id: int

class iamai.adapter.gensokyo.event.**PokeNotifyEvent**(*, *adapter: Any*, *post_type: str | None*, *time: int*, *self_id: int*, *notice_type: Literal['notify']*, *sub_type: Literal['poke']*, *group_id: int | None = None*, *user_id: int*, *target_id: int*, ***extra_data: Any*)

Bases: *NotifyEvent*

group_id: int | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'group_id': FieldInfo(annotation=Union[int, NoneType], required=False), 'notice_type': FieldInfo(annotation=Literal['notify'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=int, required=True), 'sub_type': FieldInfo(annotation=Literal['poke'], required=True), 'target_id': FieldInfo(annotation=int, required=True), 'time': FieldInfo(annotation=int, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True, alias='post_type', alias_priority=2), 'user_id': FieldInfo(annotation=int, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

sub_type: Literal['poke']

target_id: int

class iamai.adapter.gensokyo.event.**PrivateMessageEvent**(*, *adapter: Any*, *post_type: str | None*, *time: int*, *self_id: int*, *message_type: Literal['private']*, *sub_type: Literal['friend', 'group', 'other']*, *message_id: int*, *user_id: int*, *message:* GSKMessage, *raw_message: str*, *font: int*, *sender:* Sender, ***extra_data: Any*)

Bases: *MessageEvent*

message_type: Literal['private']

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'font': FieldInfo(annotation=int, required=True), 'message': FieldInfo(annotation=GSKMessage, required=True), 'message_id': FieldInfo(annotation=int, required=True), 'message_type': FieldInfo(annotation=Literal['private'], required=True), 'post_type': FieldInfo(annotation=Literal['message'], required=True), 'raw_message': FieldInfo(annotation=str, required=True), 'self_id': FieldInfo(annotation=int, required=True), 'sender': FieldInfo(annotation=Sender, required=True), 'sub_type': FieldInfo(annotation=Literal['friend', 'group', 'other'], required=True), 'time': FieldInfo(annotation=int, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True, alias='post_type', alias_priority=2), 'user_id': FieldInfo(annotation=int, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

async reply(*message: List[*GSKMessageSegment*]* | GSKMessageSegment | *str* | *Mapping[str, Any]*) → Dict[str, Any]

Parameters
message – *call_api()*

Returns
API

sub_type: Literal['friend', 'group', 'other']

class iamai.adapter.gensokyo.event.**RequestEvent**(*, *adapter: Any*, *post_type: str | None*, *time: int*, *self_id: int*, *request_type: str*, ***extra_data: Any*)

Bases: *GSKEvent*

async approve() → Dict[str, Any]

Returns
API

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'post_type': FieldInfo(annotation=Literal['request'], required=True), 'request_type': FieldInfo(annotation=str, required=True), 'self_id': FieldInfo(annotation=int, required=True), 'time': FieldInfo(annotation=int, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True, alias='post_type', alias_priority=2)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

post_type: Literal['request']

async refuse() → Dict[str, Any]

Returns
API

request_type: str

class iamai.adapter.gensokyo.event.**Sender**(*, *user_id: int | None = None*, *nickname: str | None = None*, *card: str | None = None*, *sex: Literal['male', 'female', 'unknown'] | None = None*, *age: int | None = None*, *area: str | None = None*, *level: str | None = None*, *role: str | None = None*, *title: str | None = None*)

Bases: BaseModel

age: int | None

area: str | None

card: str | None

level: str | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'age': FieldInfo(annotation=Union[int, NoneType], required=False), 'area': FieldInfo(annotation=Union[str, NoneType], required=False), 'card': FieldInfo(annotation=Union[str, NoneType], required=False), 'level': FieldInfo(annotation=Union[str, NoneType], required=False), 'nickname': FieldInfo(annotation=Union[str, NoneType], required=False), 'role': FieldInfo(annotation=Union[str, NoneType], required=False), 'sex': FieldInfo(annotation=Union[Literal['male', 'female', 'unknown'], NoneType], required=False), 'title': FieldInfo(annotation=Union[str, NoneType], required=False), 'user_id': FieldInfo(annotation=Union[int, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

nickname: str | None

role: str | None

sex: Literal['male', 'female', 'unknown'] | None

title: str | None

user_id: int | None

class iamai.adapter.gensokyo.event.Status(*, *online: bool*, *good: bool*, ***extra_data: Any*)

Bases: BaseModel

good: bool

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'good': FieldInfo(annotation=bool, required=True), 'online': FieldInfo(annotation=bool, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

online: bool

iamai.adapter.gensokyo.exceptions module

GSK

exception iamai.adapter.gensokyo.exceptions.**ActionFailed**(*resp: Dict[str, Any]*)

Bases: *GSKException*

API API

exception iamai.adapter.gensokyo.exceptions.**ApiNotAvailable**(*resp: Dict[str, Any]*)

Bases: *ActionFailed*

API 404 API

ERROR_CODE: ClassVar[int] = 1404

exception iamai.adapter.gensokyo.exceptions.**ApiTimeout**

Bases: *GSKException*

API

exception iamai.adapter.gensokyo.exceptions.**GSKException**

Bases: *AdapterException*

GSK

exception iamai.adapter.gensokyo.exceptions.**NetworkError**

Bases: *GSKException*

iamai.adapter.gensokyo.message module

GSK

class iamai.adapter.gensokyo.message.**GSKMessage**(**messages: List[MessageSegmentT] | MessageSegmentT | str | Mapping[str, Any]*)

Bases: *Message*[GSKMessageSegment]

GSK

classmethod get_segment_class() → Type[*GSKMessageSegment*]

Returns

class iamai.adapter.gensokyo.message.**GSKMessageSegment**(***, *type: str*, *data: Dict[str, Any] = None*)

Bases: MessageSegment[GSKMessage]

GSK

classmethod anonymous(*ignore: bool | None = None*) → Self

classmethod at(*qq: int | Literal['all']*) → Self

@

classmethod contact(*type_: Literal['qq', 'group']*, *id_: int*) → Self

/

classmethod contact_friend(*id_: int*) → Self

classmethod contact_group(*id_: int*) → Self

classmethod dice() → Self

classmethod face(*id_: int*) → Self

QQ

classmethod from_str(*msg: str*) → Self

str

Parameters
msg –

Returns
str

get_cqcode() → str

CQ

Returns
CQ

classmethod get_message_class() → Type[*GSKMessage*]

Returns

classmethod image(*file: str*, *type_: Literal['flash'] | None = None*, *cache: bool = True*, *proxy: bool = True*, *timeout: int | None = None*) → Self

classmethod json_message(*data: str*) → Self

JSON

classmethod location(*lat: float*, *lon: float*, *title: str | None*, *content: str | None = None*) → Self

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'data': FieldInfo(annotation=Dict[str, Any], required=False, default_factory=dict), 'type': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

classmethod music(*type_: Literal['qq', '163', 'xm']*, *id_: int*) → Self

classmethod music_custom(*url: str*, *audio: str*, *title: str*, *content: str | None = None*, *image: str | None = None*) → Self

classmethod node(*id_: int*) → Self

classmethod node_custom(*user_id: int*, *nickname: str*, *content:* GSKMessage) → Self

classmethod poke(*type_: str*, *id_: int*) → Self

classmethod record(*file: str*, *magic: bool = False*, *cache: bool = True*, *proxy: bool = True*, *timeout: int | None = None*) → Self

classmethod reply(*id_: int*) → Self

classmethod rps() → Self

classmethod shake() → Self

()

classmethod share(*url: str*, *title: str*, *content: str | None = None*, *image: str | None = None*) → Self

classmethod text(*text: str*) → Self

classmethod video(*file: str*, *cache: bool = True*, *proxy: bool = True*, *timeout: int | None = None*) → Self

classmethod xml_message(*data: str*) → Self

XML

iamai.adapter.gensokyo.message.**escape**(*string: str*, *, *escape_comma: bool = True*) → str

CQ

Parameters

- **string** –
- **escape_comma** – ,

Returns

Module contents

gensokyo *ob11

gensokyo obv11 [OneBot](https://github.com/howmanybots/onebot/blob/master/README.md)

class iamai.adapter.gensokyo.**GSKAdapter**(*bot:* Bot)

Bases: *WebSocketAdapter*[*GSKEvent*, *Config*]

GSK

class Config(*, *adapter_type: Literal['ws', 'reverse-ws', 'ws-reverse'] = 'reverse-ws'*, *host: str = '127.0.0.1'*, *port: int = 8080*, *url: str = '/gsk/ws'*, *reconnect_interval: int = 3*, *api_timeout: int = 1000*, *app_id: str = ''*, *app_secret: str = ''*, *token: str = ''*, *access_token: str = ''*, ***extra_data: Any*)

Bases: *ConfigModel*

GSK

adapter_type

Type
Literal[‘ws’, ‘reverse-ws’, ‘ws-reverse’]

host

Type
str

port

Type
int

url

WebSocket

Type
str

reconnect_interval

Type
int

api_timeout

API

Type
int

access_token

Type
str

access_token: str

adapter_type: Literal['ws', 'reverse-ws', 'ws-reverse']

api_timeout: int

app_id: str

app_secret: str

host: str

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'access_token': FieldInfo(annotation=str, required=False, default=''), 'adapter_type': FieldInfo(annotation=Literal['ws', 'reverse-ws', 'ws-reverse'], required=False, default='reverse-ws'), 'api_timeout': FieldInfo(annotation=int, required=False, default=1000), 'app_id': FieldInfo(annotation=str, required=False, default=''), 'app_secret': FieldInfo(annotation=str, required=False, default=''), 'host': FieldInfo(annotation=str, required=False, default='127.0.0.1'), 'port': FieldInfo(annotation=int, required=False, default=8080), 'reconnect_interval': FieldInfo(annotation=int, required=False, default=3), 'token': FieldInfo(annotation=str, required=False, default=''), 'url': FieldInfo(annotation=str, required=False, default='/gsk/ws')}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

port: int

reconnect_interval: int

token: str

url: str

classmethod add_event_model(*event_model: Type[GSKEvent]*) → None

GSKEvent

Parameters

event_model –

async call_api(*api: str*, ***params: Any*) → Any

GSK API API

Parameters

- **api** – API
- ****params** – API

Returns

API data

Raises

- ***NetworkError*** –
- ***ApiNotAvailable*** – API 404 API
- ***ActionFailed*** – API failed API
- ***ApiTimeout*** – API

event_models: ClassVar[Dict[Tuple[str | None, str | None, str | None], Type[*GSKEvent*]]] = {('message', 'group', None): <class 'iamai.adapter.gensokyo.event.GroupMessageEvent'>, ('message', 'private', None): <class 'iamai.adapter.gensokyo.event.PrivateMessageEvent'>, ('message', None, None): <class 'iamai.adapter.gensokyo.event.MessageEvent'>, ('meta_event', 'heartbeat', None): <class 'iamai.adapter.gensokyo.event.HeartbeatMetaEvent'>, ('meta_event', 'lifecycle', None): <class 'iamai.adapter.gensokyo.event.LifecycleMetaEvent'>, ('meta_event', None, None): <class 'iamai.adapter.gensokyo.event.MetaEvent'>, ('notice', 'friend_add', None): <class 'iamai.adapter.gensokyo.event.FriendAddNoticeEvent'>, ('notice', 'friend_recall', None): <class 'iamai.adapter.gensokyo.event.FriendRecallNoticeEvent'>, ('notice', 'group_admin', None): <class 'iamai.adapter.gensokyo.event.GroupAdminNoticeEvent'>, ('notice', 'group_ban', None): <class 'iamai.adapter.gensokyo.event.GroupBanNoticeEvent'>, ('notice', 'group_decrease', None): <class 'iamai.adapter.gensokyo.event.GroupDecreaseNoticeEvent'>, ('notice', 'group_increase', None): <class 'iamai.adapter.gensokyo.event.GroupIncreaseNoticeEvent'>, ('notice', 'group_recall', None): <class 'iamai.adapter.gensokyo.event.GroupRecallNoticeEvent'>, ('notice', 'group_upload', None): <class 'iamai.adapter.gensokyo.event.GroupUploadNoticeEvent'>, ('notice', 'notify', 'honor'): <class 'iamai.adapter.gensokyo.event.GroupHonorNotifyEvent'>, ('notice', 'notify', 'lucky_king'): <class 'iamai.adapter.gensokyo.event.GroupLuckyKingNotifyEvent'>, ('notice', 'notify', 'poke'): <class 'iamai.adapter.gensokyo.event.PokeNotifyEvent'>, ('notice', 'notify', None): <class 'iamai.adapter.gensokyo.event.NotifyEvent'>, ('notice', None, None): <class 'iamai.adapter.gensokyo.event.NoticeEvent'>, ('request', 'friend', None): <class 'iamai.adapter.gensokyo.event.FriendRequestEvent'>, ('request', 'group', None): <class 'iamai.adapter.gensokyo.event.GroupRequestEvent'>, ('request', None, None): <class 'iamai.adapter.gensokyo.event.RequestEvent'>, (None, None, None): <class 'iamai.adapter.gensokyo.event.GSKEvent'>}

async get_access_token() → str

https://bots.qq.com/app/getAppAccessToken appId string clientSecret string

classmethod get_event_model(*post_type: str | None*, *detail_type: str | None*, *sub_type: str | None*) → Type[*GSKEvent*]

Parameters

- **post_type** –
- **detail_type** –
- **sub_type** –

Returns

async handle_gsk_event(*msg: Dict[str, Any]*) → None

GSK

Parameters

msg –

async handle_websocket_msg(*msg: WSMessage*) → None

WebSocket

name: `str = 'gensokyo'`

async reverse_ws_connection_hook() → None

WebSocket

async send(*message_: List[GSKMessageSegment] | GSKMessageSegment | str | Mapping[str, Any]*, *message_type: Literal['private', 'group']*, *id_: int*) → Any

send_private_msg *send_group_msg* API

Parameters

- **message** – *str*, *Mapping*, *Iterable[Mapping]*, *GSKMessageSegment*, *GSKMessage* *GSKMessage*
- **message_type** – "private" "group"
- **id** – ID QQ

Returns

API

Raises

- **TypeError** – *message_type* "private" "group"
- **...** – *call_api()*

async startup() → None

async websocket_connect() → None

WebSocket

iamai.adapter.kook package

Subpackages

iamai.adapter.kook.api package

Submodules

iamai.adapter.kook.api.client module

class iamai.adapter.kook.api.client.**ApiClient**

Bases: `object`

iamai.adapter.kook.api.handle module

iamai.adapter.kook.api.handle.**get_api_method**(*api: str*) → str

iamai.adapter.kook.api.handle.**get_api_restype**(*api: str*) → Any

iamai.adapter.kook.api.model module

class iamai.adapter.kook.api.model.**Attachments**(*, *type: str | None = None*, *url: str | None = None*, *name: str | None = None*, *size: int | None = None*)

Bases: BaseModel

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'name': FieldInfo(annotation=Union[str, NoneType], required=False), 'size': FieldInfo(annotation=Union[int, NoneType], required=False), 'type': FieldInfo(annotation=Union[str, NoneType], required=False), 'url': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

name: str | None

size: int | None

B

type: str | None

url: str | None

class iamai.adapter.kook.api.model.**BaseMessage**(*, *id: str | None = None*, *type: int | None = None*, *content: str | None = None*, *embeds: List[dict] | None = None*, *attachments: bool | Attachments | None = None*, *create_at: int | None = None*, *updated_at: int | None = None*, *reactions: List[Reaction] | None = None*, *image_name: str | None = None*, *read_status: bool | None = None*, *quote: Quote | None = None*, *mention_info: MentionInfo | None = None*)

Bases: BaseModel

attachments: bool | *Attachments* | None

content: str | None

create_at: int | None

embeds: List[dict] | None

id_: str | None

ID

image_name: str | None

mention_info: *MentionInfo* | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'attachments': FieldInfo(annotation=Union[bool, Attachments, NoneType], required=False), 'content': FieldInfo(annotation=Union[str, NoneType], required=False), 'create_at': FieldInfo(annotation=Union[int, NoneType], required=False), 'embeds': FieldInfo(annotation=Union[List[dict], NoneType], required=False), 'id_': FieldInfo(annotation=Union[str, NoneType], required=False, alias='id', alias_priority=2), 'image_name': FieldInfo(annotation=Union[str, NoneType], required=False), 'mention_info': FieldInfo(annotation=Union[MentionInfo, NoneType], required=False), 'quote': FieldInfo(annotation=Union[Quote, NoneType], required=False), 'reactions': FieldInfo(annotation=Union[List[Reaction], NoneType], required=False), 'read_status': FieldInfo(annotation=Union[bool, NoneType], required=False), 'type': FieldInfo(annotation=Union[int, NoneType], required=False), 'updated_at': FieldInfo(annotation=Union[int, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

quote: *Quote* | None

reactions: List[*Reaction*] | None

read_status: bool | None

type: int | None

updated_at: int | None

class iamai.adapter.kook.api.model.**BlackList**(*, *user_id: str | None = None*, *created_time: int | None = None*, *remark: str | None = None*, *user:* User | *None = None*)

Bases: BaseModel

created_time: int | None

()

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: `ClassVar[dict[str, FieldInfo]] = {'created_time': FieldInfo(annotation=Union[int, NoneType], required=False), 'remark': FieldInfo(annotation=Union[str, NoneType], required=False), 'user': FieldInfo(annotation=Union[User, NoneType], required=False), 'user_id': FieldInfo(annotation=Union[str, NoneType], required=False)}`

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

remark: `str | None`

user: *User* `| None`

user_id: `str | None`

id

class iamai.adapter.kook.api.model.**BlackListsReturn**(*, *meta:* Meta | *None* = *None*, *sort: Dict[str, Any]* | *None* = *None*, *items: List[*BlackList*]* | *None* = *None*)

Bases: *ListReturn*

blacklists: `List[BlackList] | None`

model_computed_fields: `ClassVar[dict[str, ComputedFieldInfo]] = {}`

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: `ClassVar[ConfigDict] = {}`

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: `ClassVar[dict[str, FieldInfo]] = {'blacklists': FieldInfo(annotation=Union[List[BlackList], NoneType], required=False, alias='items', alias_priority=2), 'meta': FieldInfo(annotation=Union[Meta, NoneType], required=False), 'sort': FieldInfo(annotation=Union[Dict[str, Any], NoneType], required=False)}`

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class iamai.adapter.kook.api.model.**Channel**(*, *permission_overwrites: List[*PermissionOverwrite*]* | *None* = *None*, *permission_users: List[*PermissionUser*]* | *None* = *None*, *permission_sync: int* | *None* = *None*, *id: str* | *None* = *None*, *name: str* | *None* = *None*, *user_id: str* | *None* = *None*, *master_id: str* | *None* = *None*, *guild_id: str* | *None* = *None*, *topic: str* | *None* = *None*, *is_category: bool* | *None* = *None*, *parent_id: str* | *None* = *None*, *level: int* | *None* = *None*, *slow_mode: int* | *None* = *None*, *type: int* | *None* = *None*, *has_password: bool* | *None* = *None*, *limit_amount: int* | *None* = *None*)

Bases: *ChannelRoleInfo*

guild_id: str | None

id

has_password: bool | None

id_: str | None

id

is_category: bool | None

int

level: int | None

level

limit_amount: int | None

master_id: str | None

master id

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'guild_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'has_password': FieldInfo(annotation=Union[bool, NoneType], required=False), 'id_': FieldInfo(annotation=Union[str, NoneType], required=False, alias='id', alias_priority=2), 'is_category': FieldInfo(annotation=Union[bool, NoneType], required=False), 'level': FieldInfo(annotation=Union[int, NoneType], required=False), 'limit_amount': FieldInfo(annotation=Union[int, NoneType], required=False), 'master_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'name': FieldInfo(annotation=Union[str, NoneType], required=False), 'parent_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'permission_overwrites': FieldInfo(annotation=Union[List[PermissionOverwrite], NoneType], required=False), 'permission_sync': FieldInfo(annotation=Union[int, NoneType], required=False), 'permission_users': FieldInfo(annotation=Union[List[PermissionUser], NoneType], required=False), 'slow_mode': FieldInfo(annotation=Union[int, NoneType], required=False), 'topic': FieldInfo(annotation=Union[str, NoneType], required=False), 'type': FieldInfo(annotation=Union[int, NoneType], required=False), 'user_id': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

name: str | None

parent_id: str | None

id

slow_mode: int | None

, (s)

topic: `str | None`

type: `int | None`

1 , 2

Type

user_id: `str | None`

id

class iamai.adapter.kook.api.model.**ChannelMessage**(*, *id: str | None = None*, *type: int | None = None*, *content: str | None = None*, *embeds: List[dict] | None = None*, *attachments: bool | Attachments | None = None*, *create_at: int | None = None*, *updated_at: int | None = None*, *reactions: List[Reaction] | None = None*, *image_name: str | None = None*, *read_status: bool | None = None*, *quote: Quote | None = None*, *mention_info: MentionInfo | None = None*, *author: User | None = None*, *mention: List[Any] | None = None*, *mention_all: bool | None = None*, *mention_roles: List[Any] | None = None*, *mention_here: bool | None = None*)

Bases: *BaseMessage*

author: *User* | None

mention: List[Any] | None

mention_all: bool | None

mention_here: bool | None

mention_roles: List[Any] | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'attachments': FieldInfo(annotation=Union[bool, Attachments, NoneType], required=False), 'author': FieldInfo(annotation=Union[User, NoneType], required=False), 'content': FieldInfo(annotation=Union[str, NoneType], required=False), 'create_at': FieldInfo(annotation=Union[int, NoneType], required=False), 'embeds': FieldInfo(annotation=Union[List[dict], NoneType], required=False), 'id_': FieldInfo(annotation=Union[str, NoneType], required=False, alias='id', alias_priority=2), 'image_name': FieldInfo(annotation=Union[str, NoneType], required=False), 'mention': FieldInfo(annotation=Union[List[Any], NoneType], required=False), 'mention_all': FieldInfo(annotation=Union[bool, NoneType], required=False), 'mention_here': FieldInfo(annotation=Union[bool, NoneType], required=False), 'mention_info': FieldInfo(annotation=Union[MentionInfo, NoneType], required=False), 'mention_roles': FieldInfo(annotation=Union[List[Any], NoneType], required=False), 'quote': FieldInfo(annotation=Union[Quote, NoneType], required=False), 'reactions': FieldInfo(annotation=Union[List[Reaction], NoneType], required=False), 'read_status': FieldInfo(annotation=Union[bool, NoneType], required=False), 'type': FieldInfo(annotation=Union[int, NoneType], required=False), 'updated_at': FieldInfo(annotation=Union[int, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class iamai.adapter.kook.api.model.**ChannelMessagesReturn**(*, *items: List[ChannelMessage] | None = None*)

Bases: BaseModel

direct_messages: List[*ChannelMessage*] | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'direct_messages': FieldInfo(annotation=Union[List[ChannelMessage], NoneType], required=False, alias='items', alias_priority=2)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class iamai.adapter.kook.api.model.**ChannelRoleInfo**(*, *permission_overwrites: List[PermissionOverwrite] | None = None*, *permission_users: List[PermissionUser] | None = None*, *permission_sync: int | None = None*)

Bases: BaseModel

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'permission_overwrites': FieldInfo(annotation=Union[List[PermissionOverwrite], NoneType], required=False), 'permission_sync': FieldInfo(annotation=Union[int, NoneType], required=False), 'permission_users': FieldInfo(annotation=Union[List[PermissionUser], NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

permission_overwrites: List[*PermissionOverwrite*] | None

permission_sync: int | None

, 1 or 0

permission_users: List[*PermissionUser*] | None

class iamai.adapter.kook.api.model.**ChannelRoleReturn**(*, *role_id: int | None = None*, *user_id: str | None = None*, *allow: int | None = None*, *deny: int | None = None*)

Bases: BaseModel

allow: int | None

deny: int | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'allow': FieldInfo(annotation=Union[int, NoneType], required=False), 'deny': FieldInfo(annotation=Union[int, NoneType], required=False), 'role_id': FieldInfo(annotation=Union[int, NoneType], required=False), 'user_id': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

role_id: int | None

user_id: str | None

class iamai.adapter.kook.api.model.**ChannelsReturn**(*, *meta: Meta | None = None*, *sort: Dict[str, Any] | None = None*, *items: List[Channel] | None = None*)

Bases: *ListReturn*

channels: List[*Channel*] | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'channels': FieldInfo(annotation=Union[List[Channel], NoneType], required=False, alias='items', alias_priority=2), 'meta': FieldInfo(annotation=Union[Meta, NoneType], required=False), 'sort': FieldInfo(annotation=Union[Dict[str, Any], NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class iamai.adapter.kook.api.model.**DirectMessage**(*, *id: str | None = None*, *type: int | None = None*, *content: str | None = None*, *embeds: List[dict] | None = None*, *attachments: bool | Attachments | None = None*, *create_at: int | None = None*, *updated_at: int | None = None*, *reactions: List[Reaction] | None = None*, *image_name: str | None = None*, *read_status: bool | None = None*, *quote: Quote | None = None*, *mention_info: MentionInfo | None = None*, *author_id: str | None = None*, *from_type: int | None = None*, *msg_icon: str | None = None*)

Bases: *BaseMessage*

author_id: str | None

ID

from_type: int | None

from_type

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'attachments': FieldInfo(annotation=Union[bool, Attachments, NoneType], required=False), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'content': FieldInfo(annotation=Union[str, NoneType], required=False), 'create_at': FieldInfo(annotation=Union[int, NoneType], required=False), 'embeds': FieldInfo(annotation=Union[List[dict], NoneType], required=False), 'from_type': FieldInfo(annotation=Union[int, NoneType], required=False), 'id_': FieldInfo(annotation=Union[str, NoneType], required=False, alias='id', alias_priority=2), 'image_name': FieldInfo(annotation=Union[str, NoneType], required=False), 'mention_info': FieldInfo(annotation=Union[MentionInfo, NoneType], required=False), 'msg_icon': FieldInfo(annotation=Union[str, NoneType], required=False), 'quote': FieldInfo(annotation=Union[Quote, NoneType], required=False), 'reactions': FieldInfo(annotation=Union[List[Reaction], NoneType], required=False), 'read_status': FieldInfo(annotation=Union[bool, NoneType], required=False), 'type': FieldInfo(annotation=Union[int, NoneType], required=False), 'updated_at': FieldInfo(annotation=Union[int, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

msg_icon: str | None

msg_icon

class iamai.adapter.kook.api.model.**DirectMessagesReturn**(*, *items: List[*DirectMessage*] | None = None*)

Bases: BaseModel

direct_messages: List[*DirectMessage*] | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'direct_messages': FieldInfo(annotation=Union[List[DirectMessage], NoneType], required=False, alias='items', alias_priority=2)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class iamai.adapter.kook.api.model.**Emoji**(*, *id: str | None = None*, *name: str | None = None*)

Bases: BaseModel

id_: str | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'id_': FieldInfo(annotation=Union[str, NoneType], required=False, alias='id', alias_priority=2), 'name': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

name: str | None

class iamai.adapter.kook.api.model.**GuilRoleReturn**(*, *user_id: str | None = None*, *guild_id: str | None = None*, *roles: List[int] | None = None*)

Bases: BaseModel

guild_id: str | None

id

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'guild_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'roles': FieldInfo(annotation=Union[List[int], NoneType], required=False), 'user_id': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

roles: List[int] | None

id

user_id: str | None

id

class iamai.adapter.kook.api.model.**Guild**(*, *id: str | None = None*, *name: str | None = None*, *topic: str | None = None*, *user_id: str | None = None*, *icon: str | None = None*, *notify_type: int | None = None*, *region: str | None = None*, *enable_open: bool | None = None*, *open_id: str | None = None*, *default_channel_id: str | None = None*, *welcome_channel_id: str | None = None*, *roles: List[*Role*] | None = None*, *channels: List[*Channel*] | None = None*)

Bases: BaseModel

channels: List[*Channel*] | None

default_channel_id: str | None

id

enable_open: bool | None

icon: str | None

icon

id_: str | None

id

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'channels': FieldInfo(annotation=Union[List[Channel], NoneType], required=False), 'default_channel_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'enable_open': FieldInfo(annotation=Union[bool, NoneType], required=False), 'icon': FieldInfo(annotation=Union[str, NoneType], required=False), 'id_': FieldInfo(annotation=Union[str, NoneType], required=False, alias='id', alias_priority=2), 'name': FieldInfo(annotation=Union[str, NoneType], required=False), 'notify_type': FieldInfo(annotation=Union[int, NoneType], required=False), 'open_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'region': FieldInfo(annotation=Union[str, NoneType], required=False), 'roles': FieldInfo(annotation=Union[List[Role], NoneType], required=False), 'topic': FieldInfo(annotation=Union[str, NoneType], required=False), 'user_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'welcome_channel_id': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

name: str | None

notify_type: int | None

`0`

`1`

`2`@

`3`

open_id: str | None

id

region: str | None

roles: List[*Role*] | None

topic: str | None

user_id: str | None

id

welcome_channel_id: str | None

id

class iamai.adapter.kook.api.model.**GuildEmoji**(*, *name: str | None = None*, *id: str | None = None*, *user_info:* User *| None = None*)

Bases: BaseModel

id_: str | None

ID

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'id_': FieldInfo(annotation=Union[str, NoneType], required=False, alias='id', alias_priority=2), 'name': FieldInfo(annotation=Union[str, NoneType], required=False), 'user_info': FieldInfo(annotation=Union[User, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

name: str | None

user_info: *User* | None

class iamai.adapter.kook.api.model.**GuildEmojisReturn**(*, *meta:* Meta *| None = None*, *sort: Dict[str, Any] | None = None*, *items: List[*GuildEmoji*] | None = None*)

Bases: *ListReturn*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'meta': FieldInfo(annotation=Union[Meta, NoneType], required=False), 'roles': FieldInfo(annotation=Union[List[GuildEmoji], NoneType], required=False, alias='items', alias_priority=2), 'sort': FieldInfo(annotation=Union[Dict[str, Any], NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

roles: List[*GuildEmoji*] | None

class iamai.adapter.kook.api.model.**GuildUsersRetrun**(*, *meta:* Meta | *None* = *None*, *sort: Dict[str, Any]* | *None* = *None*, *items: List[*User*]* | *None* = *None*, *user_count: int* | *None* = *None*, *online_count: int* | *None* = *None*, *offline_count: int* | *None* = *None*)

Bases: *ListReturn*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'meta': FieldInfo(annotation=Union[Meta, NoneType], required=False), 'offline_count': FieldInfo(annotation=Union[int, NoneType], required=False), 'online_count': FieldInfo(annotation=Union[int, NoneType], required=False), 'sort': FieldInfo(annotation=Union[Dict[str, Any], NoneType], required=False), 'user_count': FieldInfo(annotation=Union[int, NoneType], required=False), 'users': FieldInfo(annotation=Union[List[User], NoneType], required=False, alias='items', alias_priority=2)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

offline_count: int | None

online_count: int | None

user_count: int | None

users: List[*User*] | None

class iamai.adapter.kook.api.model.**GuildsReturn**(*, *meta:* Meta | *None* = *None*, *sort: Dict[str, Any]* | *None* = *None*, *items: List[*Guild*]* | *None* = *None*)

Bases: *ListReturn*

guilds: List[*Guild*] | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'guilds': FieldInfo(annotation=Union[List[Guild], NoneType], required=False, alias='items', alias_priority=2), 'meta': FieldInfo(annotation=Union[Meta, NoneType], required=False), 'sort': FieldInfo(annotation=Union[Dict[str, Any], NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class iamai.adapter.kook.api.model.**IntimacyImg**(*, *id: str | None = None*, *url: str | None = None*)

Bases: BaseModel

id_: str | None

id

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'id_': FieldInfo(annotation=Union[str, NoneType], required=False, alias='id', alias_priority=2), 'url': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

url: str | None

class iamai.adapter.kook.api.model.**IntimacyIndexReturn**(*, *img_url: str | None = None*, *social_info: str | None = None*, *last_read: int | None = None*, *score: int | None = None*, *img_list: List[*IntimacyImg*] | None = None*)

Bases: BaseModel

img_list: List[*IntimacyImg*] | None

img_url: str | None

last_read: int | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'img_list': FieldInfo(annotation=Union[List[IntimacyImg], NoneType], required=False), 'img_url': FieldInfo(annotation=Union[str, NoneType], required=False), 'last_read': FieldInfo(annotation=Union[int, NoneType], required=False), 'score': FieldInfo(annotation=Union[int, NoneType], required=False), 'social_info': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

score: int | None

0-2200

social_info: str | None

class iamai.adapter.kook.api.model.**Invite**(*, *guild_id: str | None = None*, *channel_id: str | None = None*, *url_code: str | None = None*, *url: str | None = None*, *user:* User | *None = None*)

Bases: BaseModel

channel_id: str | None

id

guild_id: str | None

id

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'channel_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'guild_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'url': FieldInfo(annotation=Union[str, NoneType], required=False), 'url_code': FieldInfo(annotation=Union[str, NoneType], required=False), 'user': FieldInfo(annotation=Union[User, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

url: str | None

url_code: str | None

url code

user: *User* | None

class iamai.adapter.kook.api.model.**InvitesReturn**(*, *meta:* Meta | *None = None*, *sort: Dict[str, Any] | None = None*, *items: List[*Invite*] | None = None*)

Bases: *ListReturn*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'meta': FieldInfo(annotation=Union[Meta, NoneType], required=False), 'roles': FieldInfo(annotation=Union[List[Invite], NoneType], required=False, alias='items', alias_priority=2), 'sort': FieldInfo(annotation=Union[Dict[str, Any], NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

roles: List[*Invite*] | None

class iamai.adapter.kook.api.model.**ListReturn**(*, *meta:* Meta | *None* = *None*, *sort: Dict[str, Any]* | *None* = *None*)

Bases: BaseModel

meta: *Meta* | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'meta': FieldInfo(annotation=Union[Meta, NoneType], required=False), 'sort': FieldInfo(annotation=Union[Dict[str, Any], NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

sort: Dict[str, Any] | None

class iamai.adapter.kook.api.model.**MentionInfo**(*, *mention_part: List[dict]* | *None* = *None*, *mention_role_part: List[dict]* | *None* = *None*, *channel_part: List[dict]* | *None* = *None*, *item_part: List[dict]* | *None* = *None*)

Bases: BaseModel

channel_part: List[dict] | None

item_part: List[dict] | None

mention_part: List[dict] | None

mention_role_part: List[dict] | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'channel_part': FieldInfo(annotation=Union[List[dict], NoneType], required=False), 'item_part': FieldInfo(annotation=Union[List[dict], NoneType], required=False), 'mention_part': FieldInfo(annotation=Union[List[dict], NoneType], required=False), 'mention_role_part': FieldInfo(annotation=Union[List[dict], NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class iamai.adapter.kook.api.model.**MessageCreateReturn**(*, *msg_id: str | None = None*, *msg_timestamp: int | None = None*, *nonce: str | None = None*)

Bases: BaseModel

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'msg_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'msg_timestamp': FieldInfo(annotation=Union[int, NoneType], required=False), 'nonce': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

msg_id: str | None

id

msg_timestamp: int | None

()

nonce: str | None

class iamai.adapter.kook.api.model.**Meta**(*, *page: int | None = None*, *page_total: int | None = None*, *page_size: int | None = None*, *total: int | None = None*)

Bases: BaseModel

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'page': FieldInfo(annotation=Union[int, NoneType], required=False), 'page_size': FieldInfo(annotation=Union[int, NoneType], required=False), 'page_total': FieldInfo(annotation=Union[int, NoneType], required=False), 'total': FieldInfo(annotation=Union[int, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

page: int | None

page_size: int | None

page_total: int | None

total: int | None

class iamai.adapter.kook.api.model.**PermissionOverwrite**(*, *role_id: int | None = None*, *allow: int | None = None*, *deny: int | None = None*)

Bases: BaseModel

allow: int | None

deny: int | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'allow': FieldInfo(annotation=Union[int, NoneType], required=False), 'deny': FieldInfo(annotation=Union[int, NoneType], required=False), 'role_id': FieldInfo(annotation=Union[int, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

role_id: int | None

class iamai.adapter.kook.api.model.**PermissionUser**(*, *user: User | None = None*, *allow: int | None = None*, *deny: int | None = None*)

Bases: BaseModel

allow: int | None

deny: int | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'allow': FieldInfo(annotation=Union[int, NoneType], required=False), 'deny': FieldInfo(annotation=Union[int, NoneType], required=False), 'user': FieldInfo(annotation=Union[User, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

user: *User* | None

class iamai.adapter.kook.api.model.**Quote**(*, *id: str | None = None*, *type: int | None = None*, *content: str | None = None*, *create_at: int | None = None*, *author:* User | *None = None*)

Bases: BaseModel

author: *User* | None

content: str | None

create_at: int | None

id_: str | None

id

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'author': FieldInfo(annotation=Union[User, NoneType], required=False), 'content': FieldInfo(annotation=Union[str, NoneType], required=False), 'create_at': FieldInfo(annotation=Union[int, NoneType], required=False), 'id_': FieldInfo(annotation=Union[str, NoneType], required=False, alias='id', alias_priority=2), 'type': FieldInfo(annotation=Union[int, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

type: int | None

class iamai.adapter.kook.api.model.**Reaction**(*, *emoji:* Emoji | *None = None*, *count: int | None = None*, *me: bool | None = None*)

Bases: BaseModel

count: int | None

emoji: *Emoji* | None

me: bool | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'count': FieldInfo(annotation=Union[int, NoneType], required=False), 'emoji': FieldInfo(annotation=Union[Emoji, NoneType], required=False), 'me': FieldInfo(annotation=Union[bool, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class iamai.adapter.kook.api.model.**ReactionUser**(*, *id: str | None = None*, *username: str | None = None*, *nickname: str | None = None*, *identify_num: str | None = None*, *online: bool | None = None*, *bot: bool | None = None*, *os: str | None = None*, *status: int | None = None*, *avatar: str | None = None*, *vip_avatar: str | None = None*, *mobile_verified: bool | None = None*, *roles: List[int] | None = None*, *joined_at: int | None = None*, *active_time: int | None = None*, *reaction_time: int | None = None*)

Bases: *User*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'active_time': FieldInfo(annotation=Union[int, NoneType], required=False), 'avatar': FieldInfo(annotation=Union[str, NoneType], required=False), 'bot': FieldInfo(annotation=Union[bool, NoneType], required=False), 'id_': FieldInfo(annotation=Union[str, NoneType], required=False, alias='id', alias_priority=2), 'identify_num': FieldInfo(annotation=Union[str, NoneType], required=False), 'joined_at': FieldInfo(annotation=Union[int, NoneType], required=False), 'mobile_verified': FieldInfo(annotation=Union[bool, NoneType], required=False), 'nickname': FieldInfo(annotation=Union[str, NoneType], required=False), 'online': FieldInfo(annotation=Union[bool, NoneType], required=False), 'os': FieldInfo(annotation=Union[str, NoneType], required=False), 'reaction_time': FieldInfo(annotation=Union[int, NoneType], required=False), 'roles': FieldInfo(annotation=Union[List[int], NoneType], required=False), 'status': FieldInfo(annotation=Union[int, NoneType], required=False), 'username': FieldInfo(annotation=Union[str, NoneType], required=False), 'vip_avatar': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

reaction_time: int | None

class iamai.adapter.kook.api.model.**Role**(*, *role_id: int | None = None*, *name: str | None = None*, *color: int | None = None*, *position: int | None = None*, *hoist: int | None = None*, *mentionable: int | None = None*, *permissions: int | None = None*)

Bases: BaseModel

color: int | None

hoist: int | None

()

mentionable: int | None

@

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'color': FieldInfo(annotation=Union[int, NoneType], required=False), 'hoist': FieldInfo(annotation=Union[int, NoneType], required=False), 'mentionable': FieldInfo(annotation=Union[int, NoneType], required=False), 'name': FieldInfo(annotation=Union[str, NoneType], required=False), 'permissions': FieldInfo(annotation=Union[int, NoneType], required=False), 'position': FieldInfo(annotation=Union[int, NoneType], required=False), 'role_id': FieldInfo(annotation=Union[int, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

name: str | None

permissions: int | None

position: int | None

role_id: int | None

id

class iamai.adapter.kook.api.model.**RolesReturn**(*, *meta: Meta | None = None*, *sort: Dict[str, Any] | None = None*, *items: List[Role] | None = None*)

Bases: *ListReturn*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'meta': FieldInfo(annotation=Union[Meta, NoneType], required=False), 'roles': FieldInfo(annotation=Union[List[Role], NoneType], required=False, alias='items', alias_priority=2), 'sort': FieldInfo(annotation=Union[Dict[str, Any], NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

roles: List[*Role*] | None

class iamai.adapter.kook.api.model.**TargetInfo**(*, *id: str | None = None*, *username: str | None = None*, *online: bool | None = None*, *avatar: str | None = None*)

Bases: BaseModel

avatar: str | None

id_: str | None

ID

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'avatar': FieldInfo(annotation=Union[str, NoneType], required=False), 'id_': FieldInfo(annotation=Union[str, NoneType], required=False, alias='id', alias_priority=2), 'online': FieldInfo(annotation=Union[bool, NoneType], required=False), 'username': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

online: bool | None

username: str | None

class iamai.adapter.kook.api.model.**URL**(*, *url: str | None = None*)

Bases: BaseModel

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'url': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

url: `str | None`

url

class iamai.adapter.kook.api.model.**User**(*, *id: str | None = None*, *username: str | None = None*, *nickname: str | None = None*, *identify_num: str | None = None*, *online: bool | None = None*, *bot: bool | None = None*, *os: str | None = None*, *status: int | None = None*, *avatar: str | None = None*, *vip_avatar: str | None = None*, *mobile_verified: bool | None = None*, *roles: List[int] | None = None*, *joined_at: int | None = None*, *active_time: int | None = None*)

Bases: `BaseModel`

User

https://developer.kaiheila.cn/doc/objects

active_time: `int | None`

avatar: `str | None`

url

bot: `bool | None`

id_: `str | None`

id

identify_num: `str | None`

user_name#identify_num

joined_at: `int | None`

mobile_verified: `bool | None`

model_computed_fields: `ClassVar[dict[str, ComputedFieldInfo]] = {}`

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: `ClassVar[ConfigDict] = {}`

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'active_time': FieldInfo(annotation=Union[int, NoneType], required=False), 'avatar': FieldInfo(annotation=Union[str, NoneType], required=False), 'bot': FieldInfo(annotation=Union[bool, NoneType], required=False), 'id_': FieldInfo(annotation=Union[str, NoneType], required=False, alias='id', alias_priority=2), 'identify_num': FieldInfo(annotation=Union[str, NoneType], required=False), 'joined_at': FieldInfo(annotation=Union[int, NoneType], required=False), 'mobile_verified': FieldInfo(annotation=Union[bool, NoneType], required=False), 'nickname': FieldInfo(annotation=Union[str, NoneType], required=False), 'online': FieldInfo(annotation=Union[bool, NoneType], required=False), 'os': FieldInfo(annotation=Union[str, NoneType], required=False), 'roles': FieldInfo(annotation=Union[List[int], NoneType], required=False), 'status': FieldInfo(annotation=Union[int, NoneType], required=False), 'username': FieldInfo(annotation=Union[str, NoneType], required=False), 'vip_avatar': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

nickname: str | None

online: bool | None

os: str | None

os

roles: List[int] | None

id

status: int | None

,`0``1``10`

username: str | None

vip_avatar: str | None

vip url gif

class iamai.adapter.kook.api.model.**UserChat**(*, *code: str | None = None*, *last_read_time: int | None = None*, *latest_msg_time: int | None = None*, *unread_count: int | None = None*, *target_info:* TargetInfo *| None = None*)

Bases: BaseModel

code: str | None

Code

last_read_time: int | None

()

latest_msg_time: int | None

()

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'code': FieldInfo(annotation=Union[str, NoneType], required=False), 'last_read_time': FieldInfo(annotation=Union[int, NoneType], required=False), 'latest_msg_time': FieldInfo(annotation=Union[int, NoneType], required=False), 'target_info': FieldInfo(annotation=Union[TargetInfo, NoneType], required=False), 'unread_count': FieldInfo(annotation=Union[int, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

target_info: *TargetInfo* | None

unread_count: int | None

class iamai.adapter.kook.api.model.**UserChatsReturn**(*, *meta:* Meta | *None = None*, *sort: Dict[str, Any]* | *None = None*, *items: List[*UserChat*]* | *None = None*)

Bases: *ListReturn*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'meta': FieldInfo(annotation=Union[Meta, NoneType], required=False), 'sort': FieldInfo(annotation=Union[Dict[str, Any], NoneType], required=False), 'user_chats': FieldInfo(annotation=Union[List[UserChat], NoneType], required=False, alias='items', alias_priority=2)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

user_chats: List[*UserChat*] | None

Module contents

Submodules

iamai.adapter.kook.config module

Kook

class iamai.adapter.kook.config.**Config**(*, *adapter_type: Literal['ws', 'wb'] = 'ws'*, *reconnect_interval: int = 3*, *api_timeout: int = 1000*, *access_token: str = ''*, *compress: Literal[0, 1] = 0*, *show_raw: bool = False*, *report_self_message: bool = False*, ***extra_data: Any*)

Bases: *ConfigModel*

Kook

adapter_type

Type
Literal['ws', 'wb']

reconnect_interval

Type
int

api_timeout

API

Type
int

access_token

Type
str

compress

0

Type
Literal[0, 1]

show_raw

False

Type
bool

access_token: str

adapter_type: Literal['ws', 'wb']

api_timeout: int

compress: Literal[0, 1]

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'access_token': FieldInfo(annotation=str, required=False, default=''), 'adapter_type': FieldInfo(annotation=Literal['ws', 'wb'], required=False, default='ws'), 'api_timeout': FieldInfo(annotation=int, required=False, default=1000), 'compress': FieldInfo(annotation=Literal[0, 1], required=False, default=0), 'reconnect_interval': FieldInfo(annotation=int, required=False, default=3), 'report_self_message': FieldInfo(annotation=bool, required=False, default=False), 'show_raw': FieldInfo(annotation=bool, required=False, default=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

reconnect_interval: int

report_self_message: bool

show_raw: bool

iamai.adapter.kook.event module

Kook

class iamai.adapter.kook.event.**Attachment**(*, *type: str*, *name: str*, *url: Url*, *file_type: str | None = None*, *size: int | None = None*, *duration: float | None = None*, *width: int | None = None*, *hight: int | None = None*)

Bases: BaseModel

duration: float | None

file_type: str | None

hight: int | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'duration': FieldInfo(annotation=Union[float, NoneType], required=False), 'file_type': FieldInfo(annotation=Union[str, NoneType], required=False), 'hight': FieldInfo(annotation=Union[int, NoneType], required=False), 'name': FieldInfo(annotation=str, required=True), 'size': FieldInfo(annotation=Union[int, NoneType], required=False), 'type': FieldInfo(annotation=str, required=True), 'url': FieldInfo(annotation=Url, required=True, metadata=[UrlConstraints(max_length=2083, allowed_schemes=['http', 'https'], host_required=None, default_host=None, default_port=None, default_path=None)]), 'width': FieldInfo(annotation=Union[int, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

name: str

size: int | None

type: str

url: Url

width: int | None

class iamai.adapter.kook.event.**Attachments**(*, *type: str | None = None*, *url: str | None = None*, *name: str | None = None*, *size: int | None = None*)

Bases: BaseModel

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'name': FieldInfo(annotation=Union[str, NoneType], required=False), 'size': FieldInfo(annotation=Union[int, NoneType], required=False), 'type': FieldInfo(annotation=Union[str, NoneType], required=False), 'url': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

name: str | None

size: int | None

B

type: str | None

url: str | None

class iamai.adapter.kook.event.**AttrDict**(*data=None*)

Bases: UserDict

class iamai.adapter.kook.event.**BaseMessage**(*, *id: str | None = None*, *type: int | None = None*, *content: str | None = None*, *embeds: List[dict] | None = None*, *attachments: bool | Attachments | None = None*, *create_at: int | None = None*, *updated_at: int | None = None*, *reactions: List[Reaction] | None = None*, *image_name: str | None = None*, *read_status: bool | None = None*, *quote: Quote | None = None*, *mention_info: MentionInfo | None = None*)

Bases: BaseModel

attachments: bool | *Attachments* | None

content: str | None

create_at: int | None

embeds: List[dict] | None

id_: str | None

ID

image_name: str | None

mention_info: *MentionInfo* | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'attachments': FieldInfo(annotation=Union[bool, Attachments, NoneType], required=False), 'content': FieldInfo(annotation=Union[str, NoneType], required=False), 'create_at': FieldInfo(annotation=Union[int, NoneType], required=False), 'embeds': FieldInfo(annotation=Union[List[dict], NoneType], required=False), 'id_': FieldInfo(annotation=Union[str, NoneType], required=False, alias='id', alias_priority=2), 'image_name': FieldInfo(annotation=Union[str, NoneType], required=False), 'mention_info': FieldInfo(annotation=Union[MentionInfo, NoneType], required=False), 'quote': FieldInfo(annotation=Union[Quote, NoneType], required=False), 'reactions': FieldInfo(annotation=Union[List[Reaction], NoneType], required=False), 'read_status': FieldInfo(annotation=Union[bool, NoneType], required=False), 'type': FieldInfo(annotation=Union[int, NoneType], required=False), 'updated_at': FieldInfo(annotation=Union[int, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

quote: *Quote* | None

reactions: List[*Reaction*] | None

read_status: bool | None

type: int | None

updated_at: int | None

class iamai.adapter.kook.event.**BlackList**(*, *user_id: str | None = None*, *created_time: int | None = None*, *remark: str | None = None*, *user:* User | *None = None*)

Bases: BaseModel

created_time: int | None

()

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'created_time': FieldInfo(annotation=Union[int, NoneType], required=False), 'remark': FieldInfo(annotation=Union[str, NoneType], required=False), 'user': FieldInfo(annotation=Union[User, NoneType], required=False), 'user_id': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

remark: `str | None`

user: *User* `| None`

user_id: `str | None`

id

class iamai.adapter.kook.event.**BlackListsReturn**(*, *meta:* Meta | *None* = *None*, *sort: Dict[str, Any]* | *None* = *None*, *items: List[*BlackList*]* | *None* = *None*)

Bases: *ListReturn*

blacklists: `List[BlackList] | None`

model_computed_fields: `ClassVar[dict[str, ComputedFieldInfo]] = {}`

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: `ClassVar[ConfigDict] = {}`

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: `ClassVar[dict[str, FieldInfo]] = {'blacklists': FieldInfo(annotation=Union[List[BlackList], NoneType], required=False, alias='items', alias_priority=2), 'meta': FieldInfo(annotation=Union[Meta, NoneType], required=False), 'sort': FieldInfo(annotation=Union[Dict[str, Any], NoneType], required=False)}`

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class iamai.adapter.kook.event.**CartBtnClickNoticeEvent**(*, *adapter: Any*, *type: str* | *None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str* | *None* = *None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str* | *None* = *None*, *notice_type: Literal['message_btn_click']*, *group_id: int*, ***extra_data: Any*)

Bases: *NoticeEvent*

Card Button

group_id: `int`

model_computed_fields: `ClassVar[dict[str, ComputedFieldInfo]] = {}`

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: `ClassVar[ConfigDict] = {'extra': 'allow'}`

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['message_btn_click'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['message_btn_click']

user_id: str

class iamai.adapter.kook.event.**ChannelAddReactionEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['added_reaction']*, *group_id: int*, ***extra_data: Any*)

Bases: *ChannelNoticeEvent*

reaction

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['added_reaction'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['added_reaction']

class iamai.adapter.kook.event.**ChannelAddedEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['added_channel']*, *group_id: int*, ***extra_data: Any*)

Bases: *ChannelNoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['added_channel'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['added_channel']

class iamai.adapter.kook.event.**ChannelDeleteEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['deleted_channel']*, *group_id: int*, ***extra_data: Any*)

Bases: *ChannelNoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['deleted_channel'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['deleted_channel']

class iamai.adapter.kook.event.**ChannelDeleteMessageEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['deleted_message']*, *group_id: int*, ***extra_data: Any*)

Bases: *ChannelNoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['deleted_message'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['deleted_message']

class iamai.adapter.kook.event.**ChannelDeletedReactionEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['deleted_reaction']*, *group_id: int*, ***extra_data: Any*)

Bases: *ChannelNoticeEvent*

reaction

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['deleted_reaction'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['deleted_reaction']

class iamai.adapter.kook.event.**ChannelMessage**(*, *id: str | None = None*, *type: int | None = None*, *content: str | None = None*, *embeds: List[dict] | None = None*, *attachments: bool | Attachments | None = None*, *create_at: int | None = None*, *updated_at: int | None = None*, *reactions: List[Reaction] | None = None*, *image_name: str | None = None*, *read_status: bool | None = None*, *quote: Quote | None = None*, *mention_info: MentionInfo | None = None*, *author: User | None = None*, *mention: List[Any] | None = None*, *mention_all: bool | None = None*, *mention_roles: List[Any] | None = None*, *mention_here: bool | None = None*)

Bases: *BaseMessage*

author: *User* | None

mention: List[Any] | None

mention_all: bool | None

mention_here: bool | None

mention_roles: List[Any] | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'attachments': FieldInfo(annotation=Union[bool, Attachments, NoneType], required=False), 'author': FieldInfo(annotation=Union[User, NoneType], required=False), 'content': FieldInfo(annotation=Union[str, NoneType], required=False), 'create_at': FieldInfo(annotation=Union[int, NoneType], required=False), 'embeds': FieldInfo(annotation=Union[List[dict], NoneType], required=False), 'id_': FieldInfo(annotation=Union[str, NoneType], required=False, alias='id', alias_priority=2), 'image_name': FieldInfo(annotation=Union[str, NoneType], required=False), 'mention': FieldInfo(annotation=Union[List[Any], NoneType], required=False), 'mention_all': FieldInfo(annotation=Union[bool, NoneType], required=False), 'mention_here': FieldInfo(annotation=Union[bool, NoneType], required=False), 'mention_info': FieldInfo(annotation=Union[MentionInfo, NoneType], required=False), 'mention_roles': FieldInfo(annotation=Union[List[Any], NoneType], required=False), 'quote': FieldInfo(annotation=Union[Quote, NoneType], required=False), 'reactions': FieldInfo(annotation=Union[List[Reaction], NoneType], required=False), 'read_status': FieldInfo(annotation=Union[bool, NoneType], required=False), 'type': FieldInfo(annotation=Union[int, NoneType], required=False), 'updated_at': FieldInfo(annotation=Union[int, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class iamai.adapter.kook.event.**ChannelMessageEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['message'] = 'message'*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *message_type: Literal['group']*, *sub_type: str*, *event:* EventMessage, *group_id: str*, ***extra_data: Any*)

Bases: *MessageEvent*

group_id: str

message_type: Literal['group']

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'event': FieldInfo(annotation=EventMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=str, required=True), 'message_type': FieldInfo(annotation=Literal['group'], required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'post_type': FieldInfo(annotation=Literal['message'], required=False, default='message'), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'sub_type': FieldInfo(annotation=str, required=True), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

async reply(*msg: T_KookMSG*) → Dict[str, Any]

Parameters
msg – *call_api()*

Returns
API

class iamai.adapter.kook.event.**ChannelMessagesReturn**(*, *items: List[*ChannelMessage*] | None = None*)

Bases: BaseModel

direct_messages: List[*ChannelMessage*] | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'direct_messages': FieldInfo(annotation=Union[List[ChannelMessage], NoneType], required=False, alias='items', alias_priority=2)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class iamai.adapter.kook.event.**ChannelNoticeEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: str*, *group_id: int*, ***extra_data: Any*)

Bases: *NoticeEvent*

group_id: int

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=str, required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class iamai.adapter.kook.event.**ChannelPinnedMessageEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['pinned_message']*, *group_id: int*, ***extra_data: Any*)

Bases: *ChannelNoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['pinned_message'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['pinned_message']

class iamai.adapter.kook.event.**ChannelRoleInfo**(*, *permission_overwrites: List[*PermissionOverwrite*] | None = None*, *permission_users: List[*PermissionUser*] | None = None*, *permission_sync: int | None = None*)

Bases: BaseModel

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'permission_overwrites': FieldInfo(annotation=Union[List[PermissionOverwrite], NoneType], required=False), 'permission_sync': FieldInfo(annotation=Union[int, NoneType], required=False), 'permission_users': FieldInfo(annotation=Union[List[PermissionUser], NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

permission_overwrites: List[*PermissionOverwrite*] | None

permission_sync: int | None

, 1 or 0

permission_users: List[*PermissionUser*] | None

class iamai.adapter.kook.event.**ChannelRoleReturn**(*, *role_id: int | None = None*, *user_id: str | None = None*, *allow: int | None = None*, *deny: int | None = None*)

Bases: BaseModel

allow: int | None

deny: int | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'allow': FieldInfo(annotation=Union[int, NoneType], required=False), 'deny': FieldInfo(annotation=Union[int, NoneType], required=False), 'role_id': FieldInfo(annotation=Union[int, NoneType], required=False), 'user_id': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

role_id: int | None

user_id: str | None

class iamai.adapter.kook.event.**ChannelUnpinnedMessageEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['unpinned_message']*, *group_id: int*, ***extra_data: Any*)

Bases: *ChannelNoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['unpinned_message'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['unpinned_message']

class iamai.adapter.kook.event.**ChannelUpdatedEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['updated_channel']*, *group_id: int*, ***extra_data: Any*)

Bases: *ChannelNoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['updated_channel'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['updated_channel']

class iamai.adapter.kook.event.**ChannelUpdatedMessageEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['updated_message']*, *group_id: int*, ***extra_data: Any*)

Bases: *ChannelNoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['updated_message'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['updated_message']

class iamai.adapter.kook.event.**ChannelsReturn**(*, *meta:* Meta | *None* = *None*, *sort: Dict[str, Any]* | *None* = *None*, *items: List[*Channel*]* | *None* = *None*)

Bases: *ListReturn*

channels: List[*Channel*] | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'channels': FieldInfo(annotation=Union[List[Channel], NoneType], required=False, alias='items', alias_priority=2), 'meta': FieldInfo(annotation=Union[Meta, NoneType], required=False), 'sort': FieldInfo(annotation=Union[Dict[str, Any], NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class iamai.adapter.kook.event.**DirectMessage**(*, *id: str* | *None* = *None*, *type: int* | *None* = *None*, *content: str* | *None* = *None*, *embeds: List[dict]* | *None* = *None*, *attachments: bool* | Attachments | *None* = *None*, *create_at: int* | *None* = *None*, *updated_at: int* | *None* = *None*, *reactions: List[*Reaction*]* | *None* = *None*, *image_name: str* | *None* = *None*, *read_status: bool* | *None* = *None*, *quote:* Quote | *None* = *None*, *mention_info:* MentionInfo | *None* = *None*, *author_id: str* | *None* = *None*, *from_type: int* | *None* = *None*, *msg_icon: str* | *None* = *None*)

Bases: *BaseMessage*

author_id: str | None

ID

from_type: int | None

from_type

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'attachments': FieldInfo(annotation=Union[bool, Attachments, NoneType], required=False), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'content': FieldInfo(annotation=Union[str, NoneType], required=False), 'create_at': FieldInfo(annotation=Union[int, NoneType], required=False), 'embeds': FieldInfo(annotation=Union[List[dict], NoneType], required=False), 'from_type': FieldInfo(annotation=Union[int, NoneType], required=False), 'id_': FieldInfo(annotation=Union[str, NoneType], required=False, alias='id', alias_priority=2), 'image_name': FieldInfo(annotation=Union[str, NoneType], required=False), 'mention_info': FieldInfo(annotation=Union[MentionInfo, NoneType], required=False), 'msg_icon': FieldInfo(annotation=Union[str, NoneType], required=False), 'quote': FieldInfo(annotation=Union[Quote, NoneType], required=False), 'reactions': FieldInfo(annotation=Union[List[Reaction], NoneType], required=False), 'read_status': FieldInfo(annotation=Union[bool, NoneType], required=False), 'type': FieldInfo(annotation=Union[int, NoneType], required=False), 'updated_at': FieldInfo(annotation=Union[int, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

msg_icon: str | None

msg_icon

class iamai.adapter.kook.event.**DirectMessagesReturn**(*, *items: List[*DirectMessage*] | None = None*)

Bases: BaseModel

direct_messages: List[*DirectMessage*] | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'direct_messages': FieldInfo(annotation=Union[List[DirectMessage], NoneType], required=False, alias='items', alias_priority=2)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class iamai.adapter.kook.event.**EventMessage**(*, *type: int | str*, *guild_id: str | None*, *channel_name: str | None*, *mention: List | None*, *mention_all: bool | None*, *mention_roles: List | None*, *mention_here: bool | None*, *nav_channels: List | None*, *author:* User, *kmarkdown:* Kmarkdown | *None*, *code: str | None = None*, *attachments:* Attachment | *None = None*, *content:* KookMessage)

Bases: BaseModel

attachments: *Attachment* | None

author: *User*

channel_name: str | None

code: str | None

content: *KookMessage*

guild_id: str | None

kmarkdown: *Kmarkdown* | None

mention: List | None

mention_all: bool | None

mention_here: bool | None

mention_roles: List | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'attachments': FieldInfo(annotation=Union[Attachment, NoneType], required=False), 'author': FieldInfo(annotation=User, required=True), 'channel_name': FieldInfo(annotation=Union[str, NoneType], required=True), 'code': FieldInfo(annotation=Union[str, NoneType], required=False), 'content': FieldInfo(annotation=KookMessage, required=True), 'guild_id': FieldInfo(annotation=Union[str, NoneType], required=True), 'kmarkdown': FieldInfo(annotation=Union[Kmarkdown, NoneType], required=True), 'mention': FieldInfo(annotation=Union[List, NoneType], required=True), 'mention_all': FieldInfo(annotation=Union[bool, NoneType], required=True), 'mention_here': FieldInfo(annotation=Union[bool, NoneType], required=True), 'mention_roles': FieldInfo(annotation=Union[List, NoneType], required=True), 'nav_channels': FieldInfo(annotation=Union[List, NoneType], required=True), 'type': FieldInfo(annotation=Union[int, str], required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

nav_channels: List | None

type: int | str

class iamai.adapter.kook.event.**EventTypes**(*value*)

Bases: IntEnum

Kook Kook *Kook*

audio = 8

card = 10

file = 4

image = 2

kmarkdown = 9

sys = 255

text = 1

video = 3

class iamai.adapter.kook.event.**Extra**(*, *type: int | str = None*, *guild_id: str | None = None*, *channel_name: str | None = None*, *mention: List[str] | None = None*, *mention_all: bool | None = None*, *mention_roles: List[str] | None = None*, *mention_here: bool | None = None*, *author:* User | *None = None*, *body:* AttrDict | *None = None*, *attachments:* Attachment | *None = None*, *code: str | None = None*)

Bases: BaseModel

class Config

Bases: object

arbitrary_types_allowed = True

attachments: *Attachment* | None

author: *User* | None

body: *AttrDict* | None

channel_name: str | None

code: str | None

classmethod convert_body(*v*)

guild_id: str | None

mention: List[str] | None

mention_all: bool | None

mention_here: bool | None

mention_roles: List[str] | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'attachments': FieldInfo(annotation=Union[Attachment, NoneType], required=False), 'author': FieldInfo(annotation=Union[User, NoneType], required=False), 'body': FieldInfo(annotation=Union[AttrDict, NoneType], required=False), 'channel_name': FieldInfo(annotation=Union[str, NoneType], required=False), 'code': FieldInfo(annotation=Union[str, NoneType], required=False), 'guild_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'mention': FieldInfo(annotation=Union[List[str], NoneType], required=False), 'mention_all': FieldInfo(annotation=Union[bool, NoneType], required=False), 'mention_here': FieldInfo(annotation=Union[bool, NoneType], required=False), 'mention_roles': FieldInfo(annotation=Union[List[str], NoneType], required=False), 'type_': FieldInfo(annotation=Union[int, str], required=False, alias='type', alias_priority=2)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

type_: int | str

class iamai.adapter.kook.event.**GuilRoleReturn**(*, *user_id: str | None = None*, *guild_id: str | None = None*, *roles: List[int] | None = None*)

Bases: BaseModel

guild_id: str | None

id

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'guild_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'roles': FieldInfo(annotation=Union[List[int], NoneType], required=False), 'user_id': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

roles: List[int] | None

id

user_id: str | None

id

class iamai.adapter.kook.event.**GuildAddBlockListNoticeEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['added_block_list']*, *group_id: int*, ***extra_data: Any*)

Bases: *GuildNoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['added_block_list'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['added_block_list']

class iamai.adapter.kook.event.**GuildDeleteBlockListNoticeEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['deleted_block_list']*, *group_id: int*, ***extra_data: Any*)

Bases: *GuildNoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['deleted_block_list'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['deleted_block_list']

class iamai.adapter.kook.event.**GuildDeleteNoticeEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['deleted_guild']*, *group_id: int*, ***extra_data: Any*)

Bases: *GuildNoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['deleted_guild'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['deleted_guild']

class iamai.adapter.kook.event.**GuildEmoji**(*, *name: str | None = None*, *id: str | None = None*, *user_info: User | None = None*)

Bases: BaseModel

id_: str | None

ID

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'id_': FieldInfo(annotation=Union[str, NoneType], required=False, alias='id', alias_priority=2), 'name': FieldInfo(annotation=Union[str, NoneType], required=False), 'user_info': FieldInfo(annotation=Union[User, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

name: str | None

user_info: *User* | None

class iamai.adapter.kook.event.**GuildEmojisReturn**(*, *meta:* Meta | *None* = *None*, *sort: Dict[str, Any]* | *None* = *None*, *items: List[*GuildEmoji*]* | *None* = *None*)

Bases: *ListReturn*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'meta': FieldInfo(annotation=Union[Meta, NoneType], required=False), 'roles': FieldInfo(annotation=Union[List[GuildEmoji], NoneType], required=False, alias='items', alias_priority=2), 'sort': FieldInfo(annotation=Union[Dict[str, Any], NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

roles: List[*GuildEmoji*] | None

class iamai.adapter.kook.event.**GuildMemberDecreaseNoticeEvent**(*, *adapter: Any*, *type: str* | *None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str* | *None* = *None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str* | *None* = *None*, *notice_type: Literal['exited_guild']*, *group_id: int*, ***extra_data: Any*)

Bases: *GuildMemberNoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['exited_guild'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['exited_guild']

class iamai.adapter.kook.event.**GuildMemberIncreaseNoticeEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['joined_guild']*, *group_id: int*, ***extra_data: Any*)

Bases: *GuildMemberNoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['joined_guild'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['joined_guild']

class iamai.adapter.kook.event.**GuildMemberNoticeEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: str*, *group_id: int*, ***extra_data: Any*)

Bases: *GuildNoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=str, required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class iamai.adapter.kook.event.**GuildMemberOfflineNoticeEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['guild_member_offline']*, *group_id: int*, ***extra_data: Any*)

Bases: *GuildMemberNoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['guild_member_offline'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['guild_member_offline']

class iamai.adapter.kook.event.**GuildMemberOnlineNoticeEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['guild_member_online']*, *group_id: int*, ***extra_data: Any*)

Bases: *GuildMemberNoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['guild_member_online'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['guild_member_online']

class iamai.adapter.kook.event.**GuildMemberUpdateNoticeEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['updated_guild_member']*, *group_id: int*, ***extra_data: Any*)

Bases: *GuildMemberNoticeEvent*

()

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['updated_guild_member'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['updated_guild_member']

class iamai.adapter.kook.event.**GuildNoticeEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: str*, *group_id: int*, ***extra_data: Any*)

Bases: *NoticeEvent*

get_guild_id()

group_id: int

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=str, required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class iamai.adapter.kook.event.**GuildRoleAddNoticeEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['added_role']*, *group_id: int*, ***extra_data: Any*)

Bases: *GuildRoleNoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['added_role'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['added_role']

class iamai.adapter.kook.event.**GuildRoleDeleteNoticeEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['deleted_role']*, *group_id: int*, ***extra_data: Any*)

Bases: *GuildRoleNoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['deleted_role'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['deleted_role']

class iamai.adapter.kook.event.**GuildRoleNoticeEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: str*, *group_id: int*, ***extra_data: Any*)

Bases: *GuildNoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=str, required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class iamai.adapter.kook.event.**GuildRoleUpdateNoticeEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['updated_role']*, *group_id: int*, ***extra_data: Any*)

Bases: *GuildRoleNoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['updated_role'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['updated_role']

class iamai.adapter.kook.event.**GuildUpdateNoticeEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['updated_guild']*, *group_id: int*, ***extra_data: Any*)

Bases: *GuildNoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['updated_guild'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['updated_guild']

class iamai.adapter.kook.event.**GuildUsersRetrun**(*, *meta:* Meta | *None* = *None*, *sort: Dict[str, Any]* | *None* = *None*, *items: List[*User*]* | *None* = *None*, *user_count: int* | *None* = *None*, *online_count: int* | *None* = *None*, *offline_count: int* | *None* = *None*)

Bases: *ListReturn*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'meta': FieldInfo(annotation=Union[Meta, NoneType], required=False), 'offline_count': FieldInfo(annotation=Union[int, NoneType], required=False), 'online_count': FieldInfo(annotation=Union[int, NoneType], required=False), 'sort': FieldInfo(annotation=Union[Dict[str, Any], NoneType], required=False), 'user_count': FieldInfo(annotation=Union[int, NoneType], required=False), 'users': FieldInfo(annotation=Union[List[User], NoneType], required=False, alias='items', alias_priority=2)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

offline_count: int | None

online_count: int | None

user_count: int | None

users: List[*User*] | None

class iamai.adapter.kook.event.**GuildsReturn**(*, *meta:* Meta | *None* = *None*, *sort: Dict[str, Any]* | *None* = *None*, *items: List[*Guild*]* | *None* = *None*)

Bases: *ListReturn*

guilds: List[*Guild*] | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'guilds': FieldInfo(annotation=Union[List[Guild], NoneType], required=False, alias='items', alias_priority=2), 'meta': FieldInfo(annotation=Union[Meta, NoneType], required=False), 'sort': FieldInfo(annotation=Union[Dict[str, Any], NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class iamai.adapter.kook.event.**HeartbeatMetaEvent**(*, *adapter: Any*, *type: str* | *None*, *post_type: Literal['meta_event']*, *meta_event_type: Literal['heartbeat']*, *sub_type: str*, ***extra_data: Any*)

Bases: *MetaEvent*

meta_event_type: Literal['heartbeat']

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'meta_event_type': FieldInfo(annotation=Literal['heartbeat'], required=True), 'post_type': FieldInfo(annotation=Literal['meta_event'], required=True), 'sub_type': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

sub_type: str

class iamai.adapter.kook.event.**IntimacyImg**(*, *id: str* | *None* = *None*, *url: str* | *None* = *None*)

Bases: BaseModel

id_: str | None

id

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'id_': FieldInfo(annotation=Union[str, NoneType], required=False, alias='id', alias_priority=2), 'url': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

url: str | None

class iamai.adapter.kook.event.**IntimacyIndexReturn**(*, *img_url: str | None = None*, *social_info: str | None = None*, *last_read: int | None = None*, *score: int | None = None*, *img_list: List[*IntimacyImg*] | None = None*)

Bases: BaseModel

img_list: List[*IntimacyImg*] | None

img_url: str | None

last_read: int | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'img_list': FieldInfo(annotation=Union[List[IntimacyImg], NoneType], required=False), 'img_url': FieldInfo(annotation=Union[str, NoneType], required=False), 'last_read': FieldInfo(annotation=Union[int, NoneType], required=False), 'score': FieldInfo(annotation=Union[int, NoneType], required=False), 'social_info': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

score: int | None

0-2200

social_info: str | None

class iamai.adapter.kook.event.**Invite**(*, *guild_id: str | None = None*, *channel_id: str | None = None*, *url_code: str | None = None*, *url: str | None = None*, *user:* User | *None = None*)

Bases: BaseModel

channel_id: str | None

id

guild_id: str | None

id

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'channel_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'guild_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'url': FieldInfo(annotation=Union[str, NoneType], required=False), 'url_code': FieldInfo(annotation=Union[str, NoneType], required=False), 'user': FieldInfo(annotation=Union[User, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

url: str | None

url_code: str | None

url code

user: *User* | None

class iamai.adapter.kook.event.**InvitesReturn**(*, *meta:* Meta | *None = None*, *sort: Dict[str, Any] | None = None*, *items: List[*Invite*] | None = None*)

Bases: *ListReturn*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'meta': FieldInfo(annotation=Union[Meta, NoneType], required=False), 'roles': FieldInfo(annotation=Union[List[Invite], NoneType], required=False, alias='items', alias_priority=2), 'sort': FieldInfo(annotation=Union[Dict[str, Any], NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

roles: List[*Invite*] | None

class iamai.adapter.kook.event.**Kmarkdown**(*, *raw_content: str*, *mention_part: list*, *mention_role_part: list*)

Bases: BaseModel

mention_part: list

mention_role_part: list

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'mention_part': FieldInfo(annotation=list, required=True), 'mention_role_part': FieldInfo(annotation=list, required=True), 'raw_content': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

raw_content: str

class iamai.adapter.kook.event.**KookEvent**(*, *adapter: Any*, *type: str | None*, *post_type: str*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, ***extra_data: Any*)

Bases: *OriginEvent*

d Kook Kook *Kook*

author_id: str | None

channel_type: Literal['PERSON', 'GROUP']

content: *KookMessage*

extra: *Extra*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'post_type': FieldInfo(annotation=str, required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

msg_id: str

msg_timestamp: int

nonce: str

post_type: str

self_id: str | None

target_id: str

, channel_id

channel_type GROUP type 255 guild_id

type_: int

2: 3: 4: 8: 9:KMarkdown 10:card 255:

Type
1

user_id: str

class iamai.adapter.kook.event.**LifecycleMetaEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['meta_event']*, *meta_event_type: Literal['lifecycle']*, *sub_type: str*, ***extra_data: Any*)

Bases: *MetaEvent*

meta_event_type: Literal['lifecycle']

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'meta_event_type': FieldInfo(annotation=Literal['lifecycle'], required=True), 'post_type': FieldInfo(annotation=Literal['meta_event'], required=True), 'sub_type': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

sub_type: str

class iamai.adapter.kook.event.**ListReturn**(*, *meta:* Meta | *None* = *None*, *sort: Dict[str, Any]* | *None* = *None*)

Bases: BaseModel

meta: *Meta* | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'meta': FieldInfo(annotation=Union[Meta, NoneType], required=False), 'sort': FieldInfo(annotation=Union[Dict[str, Any], NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

sort: Dict[str, Any] | None

class iamai.adapter.kook.event.**MentionInfo**(*, *mention_part: List[dict]* | *None* = *None*, *mention_role_part: List[dict]* | *None* = *None*, *channel_part: List[dict]* | *None* = *None*, *item_part: List[dict]* | *None* = *None*)

Bases: BaseModel

channel_part: List[dict] | None

item_part: List[dict] | None

mention_part: List[dict] | None

mention_role_part: List[dict] | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'channel_part': FieldInfo(annotation=Union[List[dict], NoneType], required=False), 'item_part': FieldInfo(annotation=Union[List[dict], NoneType], required=False), 'mention_part': FieldInfo(annotation=Union[List[dict], NoneType], required=False), 'mention_role_part': FieldInfo(annotation=Union[List[dict], NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class iamai.adapter.kook.event.**MessageCreateReturn**(*, *msg_id: str | None = None*, *msg_timestamp: int | None = None*, *nonce: str | None = None*)

Bases: BaseModel

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'msg_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'msg_timestamp': FieldInfo(annotation=Union[int, NoneType], required=False), 'nonce': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

msg_id: str | None

id

msg_timestamp: int | None

()

nonce: str | None

class iamai.adapter.kook.event.**MessageEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['message'] = 'message'*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *message_type: str*, *sub_type: str*, *event:* EventMessage, ***extra_data: Any*)

Bases: *KookEvent*

event: *EventMessage*

get_plain_text() → str

Returns

message_type: str

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'event': FieldInfo(annotation=EventMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'message_type': FieldInfo(annotation=str, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'post_type': FieldInfo(annotation=Literal['message'], required=False, default='message'), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'sub_type': FieldInfo(annotation=str, required=True), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

post_type: Literal['message']

async reply(*msg: T_KookMSG*) → Dict[str, Any]

Parameters
msg – *call_api()*

Returns
API

sub_type: str

class iamai.adapter.kook.event.**Meta**(*, *page: int | None = None*, *page_total: int | None = None*, *page_size: int | None = None*, *total: int | None = None*)

Bases: BaseModel

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'page': FieldInfo(annotation=Union[int, NoneType], required=False), 'page_size': FieldInfo(annotation=Union[int, NoneType], required=False), 'page_total': FieldInfo(annotation=Union[int, NoneType], required=False), 'total': FieldInfo(annotation=Union[int, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

page: int | None

page_size: int | None

page_total: int | None

total: int | None

class iamai.adapter.kook.event.**MetaEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['meta_event']*, *meta_event_type: str*, ***extra_data: Any*)

Bases: *OriginEvent*

meta_event_type: str

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'meta_event_type': FieldInfo(annotation=str, required=True), 'post_type': FieldInfo(annotation=Literal['meta_event'], required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

post_type: Literal['meta_event']

class iamai.adapter.kook.event.**NoticeEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: str*, ***extra_data: Any*)

Bases: *KookEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=str, required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: str

post_type: Literal['notice']

class iamai.adapter.kook.event.**OriginEvent**(*, *adapter: Any*, *type: str | None*, *post_type: str*, ***extra_data: Any*)

Bases: Event[KookAdapter]

EventOriginEvent

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'post_type': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

post_type: str

class iamai.adapter.kook.event.**PermissionOverwrite**(*, *role_id: int | None = None*, *allow: int | None = None*, *deny: int | None = None*)

Bases: BaseModel

allow: int | None

deny: int | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'allow': FieldInfo(annotation=Union[int, NoneType], required=False), 'deny': FieldInfo(annotation=Union[int, NoneType], required=False), 'role_id': FieldInfo(annotation=Union[int, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

role_id: int | None

class iamai.adapter.kook.event.**PermissionUser**(*, *user:* User | *None* = *None*, *allow: int* | *None* = *None*, *deny: int* | *None* = *None*)

Bases: BaseModel

allow: int | None

deny: int | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'allow': FieldInfo(annotation=Union[int, NoneType], required=False), 'deny': FieldInfo(annotation=Union[int, NoneType], required=False), 'user': FieldInfo(annotation=Union[User, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

user: *User* | None

class iamai.adapter.kook.event.**PrivateAddReactionEvent**(*, *adapter: Any*, *type: str* | *None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str* | *None* = *None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str* | *None* = *None*, *notice_type: Literal['private_added_reaction']*, ***extra_data: Any*)

Bases: *PrivateNoticeEvent*

reaction

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['private_added_reaction'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['private_added_reaction']

class iamai.adapter.kook.event.**PrivateDeleteMessageEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['deleted_private_message']*, ***extra_data: Any*)

Bases: *PrivateNoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['deleted_private_message'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['deleted_private_message']

class iamai.adapter.kook.event.**PrivateDeleteReactionEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['private_deleted_reaction']*, ***extra_data: Any*)

Bases: *PrivateNoticeEvent*

reaction

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['private_deleted_reaction'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['private_deleted_reaction']

class iamai.adapter.kook.event.**PrivateMessageEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['message'] = 'message'*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *message_type: Literal['private']*, *sub_type: str*, *event:* EventMessage, ***extra_data: Any*)

Bases: *MessageEvent*

message_type: Literal['private']

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'event': FieldInfo(annotation=EventMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'message_type': FieldInfo(annotation=Literal['private'], required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'post_type': FieldInfo(annotation=Literal['message'], required=False, default='message'), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'sub_type': FieldInfo(annotation=str, required=True), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

async reply(*msg: T_KookMSG*) → Dict[str, Any]

Parameters
msg – *call_api()*

Returns
API

class iamai.adapter.kook.event.**PrivateNoticeEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: str*, ***extra_data: Any*)

Bases: *NoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=str, required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class iamai.adapter.kook.event.**PrivateUpdateMessageEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content: KookMessage*, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra: Extra*, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['updated_private_message']*, ***extra_data: Any*)

Bases: *PrivateNoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['updated_private_message'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['updated_private_message']

class iamai.adapter.kook.event.**Quote**(*, *id: str | None = None*, *type: int | None = None*, *content: str | None = None*, *create_at: int | None = None*, *author:* User *| None = None*)

Bases: BaseModel

author: *User* **| None**

content: str | None

create_at: int | None

id_: str | None

id

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'author': FieldInfo(annotation=Union[User, NoneType], required=False), 'content': FieldInfo(annotation=Union[str, NoneType], required=False), 'create_at': FieldInfo(annotation=Union[int, NoneType], required=False), 'id_': FieldInfo(annotation=Union[str, NoneType], required=False, alias='id', alias_priority=2), 'type': FieldInfo(annotation=Union[int, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

type: int | None

class iamai.adapter.kook.event.**Reaction**(*, *emoji:* Emoji *| None = None*, *count: int | None = None*, *me: bool | None = None*)

Bases: BaseModel

count: int | None

emoji: *Emoji* **| None**

me: bool | None

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'count': FieldInfo(annotation=Union[int, NoneType], required=False), 'emoji': FieldInfo(annotation=Union[Emoji, NoneType], required=False), 'me': FieldInfo(annotation=Union[bool, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class iamai.adapter.kook.event.**ReactionUser**(*, *id: str | None = None*, *username: str | None = None*, *nickname: str | None = None*, *identify_num: str | None = None*, *online: bool | None = None*, *bot: bool | None = None*, *os: str | None = None*, *status: int | None = None*, *avatar: str | None = None*, *vip_avatar: str | None = None*, *mobile_verified: bool | None = None*, *roles: List[int] | None = None*, *joined_at: int | None = None*, *active_time: int | None = None*, *reaction_time: int | None = None*)

Bases: *User*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'active_time': FieldInfo(annotation=Union[int, NoneType], required=False), 'avatar': FieldInfo(annotation=Union[str, NoneType], required=False), 'bot': FieldInfo(annotation=Union[bool, NoneType], required=False), 'id_': FieldInfo(annotation=Union[str, NoneType], required=False, alias='id', alias_priority=2), 'identify_num': FieldInfo(annotation=Union[str, NoneType], required=False), 'joined_at': FieldInfo(annotation=Union[int, NoneType], required=False), 'mobile_verified': FieldInfo(annotation=Union[bool, NoneType], required=False), 'nickname': FieldInfo(annotation=Union[str, NoneType], required=False), 'online': FieldInfo(annotation=Union[bool, NoneType], required=False), 'os': FieldInfo(annotation=Union[str, NoneType], required=False), 'reaction_time': FieldInfo(annotation=Union[int, NoneType], required=False), 'roles': FieldInfo(annotation=Union[List[int], NoneType], required=False), 'status': FieldInfo(annotation=Union[int, NoneType], required=False), 'username': FieldInfo(annotation=Union[str, NoneType], required=False), 'vip_avatar': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

reaction_time: int | None

class iamai.adapter.kook.event.**ResultStore**

Bases: object

classmethod get_sn(*self_id: str*) → int

classmethod set_sn(*self_id: str*, *sn: int*) → None

class iamai.adapter.kook.event.**RolesReturn**(*, *meta:* Meta | *None* = *None*, *sort: Dict[str, Any]* | *None* = *None*, *items: List[*Role*]* | *None* = *None*)

Bases: *ListReturn*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'meta': FieldInfo(annotation=Union[Meta, NoneType], required=False), 'roles': FieldInfo(annotation=Union[List[Role], NoneType], required=False, alias='items', alias_priority=2), 'sort': FieldInfo(annotation=Union[Dict[str, Any], NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

roles: List[*Role*] | None

class iamai.adapter.kook.event.**SelfExitGuildNoticeEvent**(*, *adapter: Any*, *type: str* | *None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str* | *None* = *None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str* | *None* = *None*, *notice_type: Literal['self_exited_guild']*, *group_id: int*, ***extra_data: Any*)

Bases: *NoticeEvent*

,

group_id: int

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['self_exited_guild'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['self_exited_guild']

user_id: str

class iamai.adapter.kook.event.**SelfJoinGuildNoticeEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content: KookMessage*, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra: Extra*, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['self_joined_guild']*, *group_id: int*, ***extra_data: Any*)

Bases: *NoticeEvent*

,

group_id: int

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['self_joined_guild'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['self_joined_guild']

user_id: str

class iamai.adapter.kook.event.**SignalTypes**(*value*)

Bases: IntEnum

Kook Kook *Kook*

EVENT = 0

HELLO = 1

PING = 2

PONG = 3

RECONNECT = 5

RESUME = 4

RESUME_ACK = 6

SYS = 255

class iamai.adapter.kook.event.**TargetInfo**(*, *id: str | None = None*, *username: str | None = None*, *online: bool | None = None*, *avatar: str | None = None*)

Bases: BaseModel

avatar: str | None

id_: str | None

ID

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'avatar': FieldInfo(annotation=Union[str, NoneType], required=False), 'id_': FieldInfo(annotation=Union[str, NoneType], required=False, alias='id', alias_priority=2), 'online': FieldInfo(annotation=Union[bool, NoneType], required=False), 'username': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

online: bool | None

username: str | None

class iamai.adapter.kook.event.**URL**(*, *url: str | None = None*)

Bases: BaseModel

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'url': FieldInfo(annotation=Union[str, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

url: str | None

url

class iamai.adapter.kook.event.**UserChat**(*, *code: str | None = None*, *last_read_time: int | None = None*, *latest_msg_time: int | None = None*, *unread_count: int | None = None*, *target_info:* TargetInfo *| None = None*)

Bases: BaseModel

code: str | None

Code

last_read_time: int | None

()

latest_msg_time: int | None

()

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'code': FieldInfo(annotation=Union[str, NoneType], required=False), 'last_read_time': FieldInfo(annotation=Union[int, NoneType], required=False), 'latest_msg_time': FieldInfo(annotation=Union[int, NoneType], required=False), 'target_info': FieldInfo(annotation=Union[TargetInfo, NoneType], required=False), 'unread_count': FieldInfo(annotation=Union[int, NoneType], required=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

target_info: *TargetInfo* | None

unread_count: int | None

class iamai.adapter.kook.event.**UserChatsReturn**(*, *meta:* Meta | *None* = *None*, *sort: Dict[str, Any]* | *None* = *None*, *items: List[*UserChat*]* | *None* = *None*)

Bases: *ListReturn*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'meta': FieldInfo(annotation=Union[Meta, NoneType], required=False), 'sort': FieldInfo(annotation=Union[Dict[str, Any], NoneType], required=False), 'user_chats': FieldInfo(annotation=Union[List[UserChat], NoneType], required=False, alias='items', alias_priority=2)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

user_chats: List[*UserChat*] | None

class iamai.adapter.kook.event.**UserInfoUpdateNoticeEvent**(*, *adapter: Any*, *type: str* | *None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str* | *None* = *None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str* | *None* = *None*, *notice_type: Literal['user_updated']*, *group_id: int*, ***extra_data: Any*)

Bases: *UserNoticeEvent*

, :

-
- **Bot**
  a.
  b.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['user_updated'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['user_updated']

class iamai.adapter.kook.event.**UserJoinAudioChannelEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['exited_channel']*, *group_id: int*, ***extra_data: Any*)

Bases: *UserNoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['exited_channel'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['exited_channel']

class iamai.adapter.kook.event.**UserJoinAudioChannelNoticeEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: Literal['joined_channel']*, *group_id: int*, ***extra_data: Any*)

Bases: *UserNoticeEvent*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=Literal['joined_channel'], required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

notice_type: Literal['joined_channel']

class iamai.adapter.kook.event.**UserNoticeEvent**(*, *adapter: Any*, *type: str | None*, *post_type: Literal['notice']*, *channel_type: Literal['PERSON', 'GROUP']*, *target_id: str*, *author_id: str | None = None*, *content:* KookMessage, *msg_id: str*, *msg_timestamp: int*, *nonce: str*, *extra:* Extra, *user_id: str*, *self_id: str | None = None*, *notice_type: str*, *group_id: int*, ***extra_data: Any*)

Bases: *NoticeEvent*

group_id: int

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'author_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'channel_type': FieldInfo(annotation=Literal['PERSON', 'GROUP'], required=True), 'content': FieldInfo(annotation=KookMessage, required=True), 'extra': FieldInfo(annotation=Extra, required=True), 'group_id': FieldInfo(annotation=int, required=True), 'msg_id': FieldInfo(annotation=str, required=True), 'msg_timestamp': FieldInfo(annotation=int, required=True), 'nonce': FieldInfo(annotation=str, required=True), 'notice_type': FieldInfo(annotation=str, required=True), 'post_type': FieldInfo(annotation=Literal['notice'], required=True), 'self_id': FieldInfo(annotation=Union[str, NoneType], required=False), 'target_id': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True), 'type_': FieldInfo(annotation=int, required=True, alias='type', alias_priority=2), 'user_id': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

iamai.adapter.kook.event.**get_event_class**(*post_type: str*, *event_type: str*, *sub_type: str | None = None*) → Type[T_KookEvent]

Parameters

- **post_type** –
- **event_type** –
- **sub_type** –

Returns

iamai.adapter.kook.exceptions module

Kook

exception iamai.adapter.kook.exceptions.**ActionFailed**(*resp*)

Bases: *KookException*

API API

exception iamai.adapter.kook.exceptions.**ApiNotAvailable**(*resp*)

Bases: *ActionFailed*

API 404 API

exception iamai.adapter.kook.exceptions.**ApiTimeout**

Bases: *KookException*

API

exception iamai.adapter.kook.exceptions.**KookException**

Bases: *AdapterException*

Kook

exception iamai.adapter.kook.exceptions.**NetworkError**

Bases: *KookException*

exception iamai.adapter.kook.exceptions.**RateLimitException**

Bases: *KookException*

exception iamai.adapter.kook.exceptions.**ReconnectError**

Bases: *KookException*

, ,

exception iamai.adapter.kook.exceptions.**TokenError**(*msg: str | None = None*)

Bases: *KookException*

, ,

exception iamai.adapter.kook.exceptions.**UnauthorizedException**

Bases: *KookException*

exception iamai.adapter.kook.exceptions.**UnsupportedMessageOperation**(*message: str = ''*)

Bases: *KookException*

Message MessageSegment MessageSegment

exception iamai.adapter.kook.exceptions.**UnsupportedMessageType**(*message: str = ''*)

Bases: *KookException*

Bot

iamai.adapter.kook.message module

Kook

class iamai.adapter.kook.message.**KookMessage**(**messages: List[MessageSegmentT] | MessageSegmentT | str | Mapping[str, Any]*)

Bases: *Message*[KookMessageSegment]

Kook v3 Message

class iamai.adapter.kook.message.**KookMessageSegment**(**, type: str, data: Dict[str, Any] = None*)

Bases: `MessageSegment[KookMessage]`

Kook

classmethod Card(*content: Any*) → *KookMessageSegment*

@param content: KMarkdownhttps://developer.kookapp.cn/doc/cardmessage

classmethod KMarkdown(*content: str*, *raw_content: str | None = None*) → *KookMessageSegment*

KMarkdown

@param content: KMarkdownhttps://developer.kookapp.cn/doc/kmarkdown @param raw_content:

classmethod at(*user_id: str*) → *KookMessageSegment*

classmethod audio(*file_key: str*, *title: str | None = None*, *cover_file_key: str | None = None*) → *KookMessageSegment*

classmethod file(*file_key: str*, *title: str | None = None*) → *KookMessageSegment*

classmethod image(*file_key: str*) → *KookMessageSegment*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'data': FieldInfo(annotation=Dict[str, Any], required=False, default_factory=dict), 'type': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

classmethod quote(*msg_id: str*) → *KookMessageSegment*

classmethod text(*text: str*) → *KookMessageSegment*

classmethod video(*file_key: str*, *title: str | None = None*) → *KookMessageSegment*

iamai.adapter.kook.message.**escape_kmarkdown**(*content: str*)

kmarkdown

iamai.adapter.kook.message.**unescape_kmarkdown**(*content: str*)

kmarkdown

Module contents

Kook

Kook : [Kook](https://developer.kookapp.cn/)

class iamai.adapter.kook.**KookAdapter**(*bot:* Bot)

Bases: *WebSocketAdapter*[*KookEvent*, *Config*]

Kook

class Config(**, adapter_type: Literal['ws', 'wb'] = 'ws', reconnect_interval: int = 3, api_timeout: int = 1000, access_token: str = '', compress: Literal[0, 1] = 0, show_raw: bool = False, report_self_message: bool = False, **extra_data: Any*)

Bases: *ConfigModel*

Kook

adapter_type

Type
Literal[‘ws’, ‘wb’]

reconnect_interval

Type
int

api_timeout

API

Type
int

access_token

Type
str

compress

0

Type
Literal[0, 1]

show_raw

False

Type
bool

access_token: str

adapter_type: Literal['ws', 'wb']

api_timeout: int

compress: Literal[0, 1]

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'access_token': FieldInfo(annotation=str, required=False, default=''), 'adapter_type': FieldInfo(annotation=Literal['ws', 'wb'], required=False, default='ws'), 'api_timeout': FieldInfo(annotation=int, required=False, default=1000), 'compress': FieldInfo(annotation=Literal[0, 1], required=False, default=0), 'reconnect_interval': FieldInfo(annotation=int, required=False, default=3), 'report_self_message': FieldInfo(annotation=bool, required=False, default=False), 'show_raw': FieldInfo(annotation=bool, required=False, default=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

reconnect_interval: int

report_self_message: bool

show_raw: bool

`api_root = 'https://www.kookapp.cn/api/v3/'`

`async call_api`(*api: str*, ***data: dict*) → Any

`async handle_kook_event`(*data: Dict[str, Any]*)

kook

Parameters

msg –

`async handle_websocket_msg`(*msg: WSMessage*)

WebSocket

`name: str = 'kook'`

`async send`(*message_: T_KookMSG*, *message_type: Literal['GROUP', 'PERSON']*, *id_: int*) → Dict[str, Any]

message/create direct-message/create API

Parameters

- **message** – str, Mapping, Iterable[Mapping], 'KookMessageSegment', 'KookMessage' *KookMessage*
- **message_type** – GROUP PERSON
- **id** – ID Kook Kook

Returns

API

Raises

- **TypeError** – message_type 'PERSON' 'GROUP'
- **...** – *call_api()*

`async start_heartbeat`(*session: str*) → None

30s :return:

`async startup`()

`async websocket_connect`()

WebSocket

iamai.adapter.red package

Submodules

iamai.adapter.red.config module

iamai.adapter.red.event module

Module contents

Submodules

iamai.adapter.utils module

class iamai.adapter.utils.**HttpClientAdapter**(*bot:* Bot)

Bases: *PollingAdapter*[EventT, ConfigT]

HTTP

abstract async on_tick() → None

session: ClientSession

async shutdown() → None

async startup() → None

class iamai.adapter.utils.**HttpServerAdapter**(*bot:* Bot)

Bases: *Adapter*[EventT, ConfigT]

HTTP

app: Application

get_url: str

abstract async handle_response(*request: Request*) → StreamResponse

host: str

port: int

post_url: str

async run() → None

runner: AppRunner

async shutdown() → None

site: TCPSite

async startup() → None

class iamai.adapter.utils.**PollingAdapter**(*bot:* Bot)

Bases: *Adapter*[EventT, ConfigT]

create_task: bool = False

delay: float = 0.1

abstract async on_tick() → None

async run() → None

class iamai.adapter.utils.**WebSocketAdapter**(*bot:* Bot)

Bases: *Adapter*[EventT, ConfigT]

WebSocket

WebSocket

adapter_type: **Literal['ws', 'reverse-ws']**

app: **Application | None**

async handle_reverse_ws_response(*request: Request*) → WebSocketResponse

aiohttp WebSocket

async handle_websocket() → None

WebSocket

abstract async handle_websocket_msg(*msg: WSMessage*) → None

WebSocket

host: **str**

port: **int**

reconnect_interval: **int = 3**

async reverse_ws_connection_hook() → None

WebSocket

async run() → None

runner: **AppRunner | None**

session: **ClientSession | None**

async shutdown() → None

site: **TCPSite | None**

async startup() → None

url: **str**

websocket: **WebSocketResponse | ClientWebSocketResponse | None = None**

async websocket_connect() → None

WebSocket

class iamai.adapter.utils.**WebSocketClientAdapter**(*bot:* Bot)

Bases: *Adapter*[EventT, ConfigT]

WebSocket

abstract async handle_response(*msg: WSMessage*) → None

async run() → None

url: **str**

class iamai.adapter.utils.**WebSocketServerAdapter**(*bot:* Bot)

Bases: *Adapter*[EventT, ConfigT]

WebSocket

app: **Application**

async handle_response(*request: Request*) → WebSocketResponse

WebSocket

abstract async handle_ws_response(*msg: WSMessage*) → None

WebSocket

host: **str**

port: **int**

async run() → None

runner: **AppRunner**

async shutdown() → None

site: **TCPSite**

async startup() → None

url: **str**

websocket: **WebSocketResponse**

Module contents

iamai

Adapter

class iamai.adapter.**Adapter**(*bot:* Bot)

Bases: Generic[EventT, ConfigT], ABC

name

Type
str

bot

Type
Bot

Config: **Type[ConfigT]**

bot: *Bot*

property config: **ConfigT**

final async get(*func: Callable[[EventT], bool | Awaitable[bool]] | None = None*, ***, *event_type: None = None*, *max_try_times: int | None = None*, *timeout: int | float | None = None*) → EventT

final async get(*func: Callable[[_EventT], bool | Awaitable[bool]] | None = None*, ***, *event_type: Type[_EventT]*, *max_try_times: int | None = None*, *timeout: int | float | None = None*) → _EventT

Bot get() Bot get() adapter_type

Parameters

- **func** – *True None*
- **event_type** – func *None*
- **max_try_times** –
- **timeout** –

Returns
func

Raises
GetEventTimeout –

name: str

abstract async run() → None

AliceBot

final async safe_run() → None

async shutdown() → None

AliceBot *shutdown()*

async startup() → None

AliceBot *startup() run()*

iamai.models package

Subpackages

iamai.models.BM25 package

Submodules

iamai.models.BM25.config module

Module contents

class iamai.models.BM25.**BM25**(*documents*)

Bases: object

get_score(*query*)

initialize()

score(*query*)

search_top_k(*query*, *k=5*)

Module contents

12.1.2 Submodules

iamai.bot module

iamai

iamai iamai *Bot*

class iamai.bot.**Bot**(**, config_file: str | None = 'config.toml', config_dict: Dict[str, Any] | None = None, hot_reload: bool = False*)

Bases: object

iamai

Config Adapter Plugin

config

Type
iamai.config.MainConfig

should_exit

Type
asyncio.locks.Event

adapters

Type
List[*iamai.adapter.Adapter*[Any, Any]]

plugins_priority_dict

Type
Dict[int, List[Type[*iamai.plugin.Plugin*[Any, Any, Any]]]]

plugin_state

Type
Dict[str, Any]

global_state

Type
Dict[Any, Any]

adapter_run_hook(*func: Callable[[*Adapter*[Any, Any]], Awaitable[None]]*) → Callable[[*Adapter*[Any, Any]], Awaitable[None]]

Parameters
func –

Returns

adapter_shutdown_hook(*func: Callable[[*Adapter*[Any, Any]], Awaitable[None]]*) → Callable[[*Adapter*[Any, Any]], Awaitable[None]]

Parameters
func –

Returns

adapter_startup_hook(*func: Callable[[*Adapter*[Any, Any]], Awaitable[None]]*) → Callable[[*Adapter*[Any, Any]], Awaitable[None]]

Parameters
func –

Returns

adapters: List[*Adapter*[Any, Any]]

bot_exit_hook(*func: Callable[[*Bot*], Awaitable[None]]*) → Callable[[*Bot*], Awaitable[None]]

Bot

Parameters
func –

Returns

bot_run_hook(*func: Callable[[*Bot*], Awaitable[None]]*) → Callable[[*Bot*], Awaitable[None]]

Bot

Parameters
func –

Returns

config: *MainConfig*

error_or_exception(*message: str*, *exception: Exception*) → None

Bot error exception

Parameters

- **message** –
- **exception** –

event_postprocessor_hook(*func: Callable[[*Event*[Any]], Awaitable[None]]*) → Callable[[*Event*[Any]], Awaitable[None]]

Parameters
func –

Returns

event_preprocessor_hook(*func: Callable[[*Event*[Any]], Awaitable[None]]*) → Callable[[*Event*[Any]], Awaitable[None]]

Parameters
func –

Returns

async get(*func: Callable[[*Event*[Any]], bool | Awaitable[bool]] | None = None*, ***, *event_type: None = None*, *adapter_type: None = None*, *max_try_times: int | None = None*, *timeout: int | float | None = None*) → *Event*[Any]

async get(*func: Callable[[EventT], bool | Awaitable[bool]] | None = None*, ***, *event_type: None = None*, *adapter_type: Type[*Adapter*[EventT, Any]]*, *max_try_times: int | None = None*, *timeout: int | float | None = None*) → EventT

async get(*func: Callable[[EventT], bool | Awaitable[bool]] | None = None*, ***, *event_type: Type[EventT]*, *adapter_type: Type[AdapterT] | None = None*, *max_try_times: int | None = None*, *timeout: int | float | None = None*) → EventT

Parameters

- **func** – *True* *None*
- **event_type** – func *None*
- **adapter_type** – func *None*
- **max_try_times** –
- **timeout** –

Returns

func

Raises

GetEventTimeout –

get_adapter(*adapter: str*) → *Adapter*[Any, Any]

get_adapter(*adapter: Type[AdapterT]*) → AdapterT

Parameters

adapter –

Returns

Raises

LookupError –

get_plugin(*name: str*) → Type[*Plugin*[Any, Any, Any]]

Parameters

name –

Returns

Raises

LookupError –

global_state: Dict[Any, Any]

async handle_event(*current_event:* Event*[Any]*, ***, *handle_get: bool = True*, *show_log: bool = True*) → None

stop *skip*

Parameters

- **current_event** – *Event*
- **handle_get** – get *True*
- **show_log** – *True*

load_adapters(**adapters: Type[Adapter[Any, Any]] | str*) → None

Parameters
***adapters** – *Type[Adapter] str Type[Adapter] str* Python *import path.of.adapter*

load_plugins(**plugins: Type[Plugin[Any, Any, Any]] | str | Path*) → None

Parameters
***plugins** – *Type[Plugin], str pathlib.Path Type[Plugin] str* Python *import path.of.plugin*

pathlib.Path
pathlib.Path("path/of/plugin")

load_plugins_from_dirs(**dirs: Path*) → None

–

Parameters
***dirs** – *pathlib.Path("path/of/plugins/")*

plugin_state: Dict[str, Any]

property plugins: List[Type[*Plugin*[Any, Any, Any]]]

plugins_priority_dict: Dict[int, List[Type[*Plugin*[Any, Any, Any]]]]

reload_plugins() → None

restart() → None
iamai

run() → None
iamai

should_exit: Event

iamai.cli module

iamai.cli.**force_delete_package**(*package_name*, *directory*)

iamai.cli.**install_package**(*package_name*, *directory*)

iamai.cli.**search_packages_with_dependency**(*dependency_name*)

iamai.config module

iamai

iamai [pydantic](https://pydantic-docs.helpmanual.io/)

class iamai.config.**AdapterConfig**(***extra_data: Any*)
Bases: *ConfigModel*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class iamai.config.**BotConfig**(*, *plugins: Set[str] = None*, *plugin_dirs: Set[Path] = None*, *adapters: Set[str] = None*, *log:* LogConfig *= LogConfig(level='DEBUG', verbose_exception=False)*, ***extra_data: Any*)

Bases: *ConfigModel*

Bot

plugins

Bot load_plugins()

Type
Set[str]

plugin_dirs

Bot load_plugins_from_dirs()

Type
Set[pathlib.Path]

adapters

Bot load_adapters()

Type
Set[str]

log

iamai

Type
iamai.config.LogConfig

adapters: Set[str]

log: *LogConfig*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapters': FieldInfo(annotation=Set[str], required=False, default_factory=set), 'log': FieldInfo(annotation=LogConfig, required=False, default=LogConfig(level='DEBUG', verbose_exception=False)), 'plugin_dirs': FieldInfo(annotation=Set[Annotated[Path, PathType]], required=False, default_factory=set), 'plugins': FieldInfo(annotation=Set[str], required=False, default_factory=set)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

plugin_dirs: Set[Path]

plugins: Set[str]

class iamai.config.**ConfigModel**(***extra_data: Any*)

Bases: BaseModel

iamai

__config_name__

Type
str

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

class iamai.config.**LogConfig**(**, level: str | int = 'DEBUG', verbose_exception: bool = False, **extra_data: Any*)

Bases: *ConfigModel*

iamai

level

Type
str | int

verbose_exception

True Traceback

Type
bool

level: str | int

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'level': FieldInfo(annotation=Union[str, int], required=False, default='DEBUG'), 'verbose_exception': FieldInfo(annotation=bool, required=False, default=False)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

verbose_exception: bool

class iamai.config.**MainConfig**(*, *bot:* BotConfig = *BotConfig(plugins=set(), plugin_dirs=set(), adapters=set(), log=LogConfig(level='DEBUG', verbose_exception=False))*, *plugin:* PluginConfig = *PluginConfig()*, *adapter:* AdapterConfig = *AdapterConfig()*, ***extra_data: Any*)

Bases: *ConfigModel*

iamai

bot

iamai

Type
iamai.config.BotConfig

adapter: *AdapterConfig*

bot: *BotConfig*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=AdapterConfig, required=False, default=AdapterConfig()), 'bot': FieldInfo(annotation=BotConfig, required=False, default=BotConfig(plugins=set(), plugin_dirs=set(), adapters=set(), log=LogConfig(level='DEBUG', verbose_exception=False))), 'plugin': FieldInfo(annotation=PluginConfig, required=False, default=PluginConfig())}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

plugin: *PluginConfig*

class iamai.config.**PluginConfig**(***extra_data: Any*)

Bases: *ConfigModel*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

iamai.const module

iamai.dependencies module

iamai

iamai.dependencies.**Depends**(*dependency: Type[_T | AsyncContextManager[_T] | ContextManager[_T]] | Callable[[], AsyncGenerator[_T, None]] | Callable[[], Generator[_T, None, None]] | None = None*, ***, *use_cache: bool = True*) → _T

Parameters

- **dependency** –
- **use_cache** – *True*

Returns

iamai.event module

iamai

class iamai.event.**Event**(***, *adapter: Any*, *type: str | None*, ***extra_data: Any*)

Bases: ABC, BaseModel, Generic[AdapterT]

adapter

Type
Any

type

Type
str | None

__handled__

Type
bool

adapter: Any

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

type: str | None

class iamai.event.**MessageEvent**(*, *adapter: Any*, *type: str | None*, ***extra_data: Any*)

Bases: *Event*, Generic[AdapterT]

adapter: Any

async ask(*message: str*, *max_try_times: int | None = None*, *timeout: int | float | None = None*) → Self

reply() get()

Parameters

- **message** –
- **max_try_times** –
- **timeout** –

Returns

async get(*, *max_try_times: int | None = None*, *timeout: int | float | None = None*) → Self

Bot get()

Parameters

- **max_try_times** –
- **timeout** –

Returns

Raises

GetEventTimeout –

abstract get_plain_text() → str

Returns

abstract async is_same_sender(*other: Self*) → bool

Parameters

other –

Returns

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

abstract async reply(*message: str*) → Any

Parameters
message –

Returns

type: str | None

iamai.exceptions module

iamai

iamai iamai *AdapterException*

exception iamai.exceptions.**AdapterException**

Bases: *iamaiException*

exception iamai.exceptions.**EventException**

Bases: BaseException

iamai

exception iamai.exceptions.**GetEventTimeout**

Bases: *iamaiException*

get

exception iamai.exceptions.**LoadModuleError**

Bases: *iamaiException*

exception iamai.exceptions.**SkipException**

Bases: *EventException*

exception iamai.exceptions.**StopException**

Bases: *EventException*

exception iamai.exceptions.**iamaiException**

Bases: Exception

iamai

iamai.log module

iamai

iamai [loguru](https://github.com/Delgan/loguru) logger [loguru](https://github.com/Delgan/loguru)

iamai.log.**error_or_exception**(*message: str*, *exception: Exception*, *verbose: bool*)

iamai.message module

iamai

Message MessageSegment

class iamai.message.**Message**(**messages: List[MessageSegmentT] | MessageSegmentT | str | Mapping[str, Any]*)

Bases: ABC, List[MessageSegmentT]

List __init__() str, *Mapping*, *MessageSegment*, *List[MessageSegment] + += Message*, *MessageSegment get_segment_class()*

copy() → Self

Returns

endswith(*suffix: str | MessageSegmentT*, *start: SupportsIndex | None = None*, *end: SupportsIndex | None = None*) → bool

endswith()

suffix str str str endswith() suffix MessageSegment *start end*

suffix

Parameters

- **suffix** –
- **start** –
- **end** –

Returns

get_plain_text() → str

Returns

abstract classmethod get_segment_class() → Type[MessageSegmentT]

Returns

is_text() → bool

replace(*old: str*, *new: str*, *count: int = -1*) → Self

replace(*old: MessageSegmentT*, *new: MessageSegmentT | None*, *count: int = -1*) → Self

replace()

old str new str is_text() True old MessageSegment *new MessageSegment None*

None

Parameters

- **old** –
- **new** –
- **count** –

Returns

startswith(*prefix: str | MessageSegmentT*, *start: SupportsIndex | None = None*, *end: SupportsIndex | None = None*) → bool

startswith()

prefix str str str startswith() prefix MessageSegment start end

prefix

Parameters

- **prefix** –
- **start** –
- **end** –

Returns

class iamai.message.**MessageSegment**(*, *type: str*, *data: Dict[str, Any] = None*)

Bases: ABC, BaseModel, Mapping[str, Any], Generic[MessageT]

Mapping data + += Message, MessageSegment Message get_message_class()

type

Type
str

data

Type
Dict[str, Any]

data: Dict[str, Any]

classmethod from_mapping(*msg: Mapping[Any, Any]*) → Self

Mapping

Mapping

Parameters
msg –

Returns
Mapping

abstract classmethod from_str(*msg: str*) → Self

str

Parameters
msg –

Returns
str

get(*key: str*, *default: Any | None = None*) → Any

key *data* *key* *default*

abstract classmethod get_message_class() → Type[MessageT]

Returns

is_text() → bool

Returns

items() → ItemsView[str, Any]

data *((,)*)

keys() → KeysView[str]

data

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'data': FieldInfo(annotation=Dict[str, Any], required=False, default_factory=dict), 'type': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

type: str

values() → ValuesView[Any]

data

iamai.plugin module

iamai

iamai *Plugin*

class iamai.plugin.**Plugin**

Bases: ABC, Generic[EventT, StateT, ConfigT]

iamai

event

Type
iamai.typing.EventT

priority

0

Type
ClassVar[int]

block

True

Type
ClassVar[bool]

__plugin_load_type__

iamai

Type
ClassVar[*iamai.plugin.PluginLoadType*]

__plugin_file_path__

PluginLoadType.CLASS *None* Python

Type
ClassVar[str | None]

Config: Type[ConfigT]

block: ClassVar[bool] = False

property bot: *Bot*

property config: ConfigT

event: EventT = InnerDepends(Event)

abstract async handle() → None

rule() *True* iamai

property name: str

priority: ClassVar[int] = 0

abstract async rule() → bool

True

handle()

final skip() → NoReturn

property state: StateT

final stop() → NoReturn

class iamai.plugin.**PluginLoadType**(*value*)

Bases: Enum

CLASS = 'class'

DIR = 'dir'

FILE = 'file'

NAME = 'name'

iamai.typing module

iamai

iamai

iamai.utils module

iamai

class iamai.utils.**ModulePathFinder**

Bases: MetaPathFinder

iamai

find_spec(*fullname: str*, *path: Sequence[str] | None = None*, *target: module | None = None*) → ModuleSpec | None

spec

path: ClassVar[List[str]] = []

class iamai.utils.**PydanticEncoder**(**, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, sort_keys=False, indent=None, separators=None, default=None*)

Bases: JSONEncoder

pydantic.BaseModel JSONEncoder

default(*o: Any*) → Any

o

iamai.utils.**get_annotations**(*obj*, ***, *globals=None*, *locals=None*, *eval_str=False*)

Compute the annotations dict for an object.

obj may be a callable, class, or module. Passing in an object of any other type raises TypeError.

Returns a dict. get_annotations() returns a new dict every time it's called; calling it twice on the same object will return two different but equivalent dicts.

This function handles several details for you:

- If eval_str is true, values of type str will be un-stringized using eval(). This is intended for use with stringized annotations ("from __future__ import annotations").
- If obj doesn't have an annotations dict, returns an empty dict. (Functions and methods always have an annotations dict; classes, modules, and other types of callables may not.)
- Ignores inherited annotations on classes. If a class doesn't have its own annotations dict, returns an empty dict.
- All accesses to object members and dict values are done using getattr() and dict.get() for safety.
- Always, always, always returns a freshly-created dict.

eval_str controls whether or not values of type str are replaced with the result of calling eval() on those values:

- If eval_str is true, eval() is called on values of type str.
- If eval_str is false (the default), values of type str are unchanged.

globals and locals are passed in to eval(); see the documentation for eval() for more information. If either globals or locals is None, this function may replace that value with a context-specific default, contingent on type(obj):

- If obj is a module, globals defaults to obj.__dict__.
- If obj is a class, globals defaults to sys.modules[obj.__module__].__dict__ and locals defaults to the obj class namespace.
- If obj is a callable, globals defaults to obj.__globals__, although if obj is a wrapped function (using functools.update_wrapper()) it is first unwrapped.

iamai.utils.**get_classes_from_module**(*module: module*, *super_class: _TypeT*) → List[_TypeT]

Parameters

- **module** – Python
- **super_class** –

Returns

iamai.utils.**get_classes_from_module_name**(*name: str*, *super_class: _TypeT*, ***, *reload: bool = False*) → List[Tuple[_TypeT, module]]

Parameters

- **name** – Python *import*
- **super_class** –
- **reload** –

Returns

Raises
ImportError –

iamai.utils.**is_config_class**(*config_class: Any*) → TypeGuard[Type[*ConfigModel*]]

Parameters
config_class –

Returns

iamai.utils.**samefile**(*path1: str | bytes | PathLike[str] | PathLike[bytes]*, *path2: str | bytes | PathLike[str] | PathLike[bytes]*) → bool

os.path.samefile

Parameters

- **path1** – 1
- **path2** – 2

Returns

iamai.utils.**sync_ctx_manager_wrapper**(*cm: ContextManager[_T]*, ***, *to_thread: bool = False*) → AsyncGenerator[_T, None]

Parameters

- **cm** –

- **to_thread** – *False*

Returns

iamai.utils.**sync_func_wrapper**(*func: Callable[[_P], _R]*, ***, *to_thread: bool = False*) → Callable[[_P], Coroutine[None, None, _R]]

Parameters

- **func** –
- **to_thread** – *False*

Returns

iamai.utils.**wrap_get_func**(*func: Callable[[EventT], bool | Awaitable[bool]] | None*) → Callable[[EventT], Awaitable[bool]]

get()

Parameters

func – *get()*

Returns

12.1.3 Module contents

API Documentation

Comprehensive AI Toolkit for Multimodal Learning and Cross-Platform Robotics.

This Module imports the following contents from the sub-module.

class iamai.**Adapter**(*bot:* Bot)

Bases: Generic[EventT, ConfigT], ABC

name

Type
str

bot

Type
Bot

Config: Type[ConfigT]

bot: *Bot*

property config: ConfigT

final async get(*func: Callable[[EventT], bool | Awaitable[bool]] | None = None*, ***, *event_type: None = None*, *max_try_times: int | None = None*, *timeout: int | float | None = None*) → EventT

final async get(*func: Callable[[_EventT], bool | Awaitable[bool]] | None = None*, ***, *event_type: Type[_EventT]*, *max_try_times: int | None = None*, *timeout: int | float | None = None*) → _EventT

Bot get() *Bot get()* adapter_type

Parameters

- **func** – *True* *None*
- **event_type** – func *None*
- **max_try_times** –
- **timeout** –

Returns
func

Raises
GetEventTimeout –

name: str

abstract async run() → None

AliceBot

final async safe_run() → None

async shutdown() → None

AliceBot *shutdown()*

async startup() → None

AliceBot *startup()* *run()*

class iamai.Bot(*, *config_file: str | None = 'config.toml'*, *config_dict: Dict[str, Any] | None = None*, *hot_reload: bool = False*)**

Bases: object

iamai

Config *Adapter* *Plugin*

config

Type
iamai.config.MainConfig

should_exit

Type
asyncio.locks.Event

adapters

Type
List[*iamai.adapter.Adapter*[Any, Any]]

plugins_priority_dict

Type
Dict[int, List[Type[*iamai.plugin.Plugin*[Any, Any, Any]]]]

plugin_state

Type
Dict[str, Any]

global_state

Type
Dict[Any, Any]

adapter_run_hook(*func: Callable[[*Adapter*[Any, Any]], Awaitable[None]]*) → Callable[[*Adapter*[Any, Any]], Awaitable[None]]

Parameters
func –

Returns

adapter_shutdown_hook(*func: Callable[[*Adapter*[Any, Any]], Awaitable[None]]*) → Callable[[*Adapter*[Any, Any]], Awaitable[None]]

Parameters
func –

Returns

adapter_startup_hook(*func: Callable[[*Adapter*[Any, Any]], Awaitable[None]]*) → Callable[[*Adapter*[Any, Any]], Awaitable[None]]

Parameters
func –

Returns

adapters: List[*Adapter*[Any, Any]]

bot_exit_hook(*func: Callable[[*Bot*], Awaitable[None]]*) → Callable[[*Bot*], Awaitable[None]]

Bot

Parameters
func –

Returns

bot_run_hook(*func: Callable[[*Bot*], Awaitable[None]]*) → Callable[[*Bot*], Awaitable[None]]

Bot

Parameters
func –

Returns

config: *MainConfig*

error_or_exception(*message: str*, *exception: Exception*) → None

Bot error exception

Parameters

- **message** –
- **exception** –

event_postprocessor_hook(*func: Callable[[*Event*[Any]], Awaitable[None]]*) → Callable[[*Event*[Any]], Awaitable[None]]

Parameters
func –

Returns

event_preprocessor_hook(*func: Callable[[*Event*[Any]], Awaitable[None]]*) → Callable[[*Event*[Any]], Awaitable[None]]

Parameters
func –

Returns

async get(*func: Callable[[*Event*[Any]], bool | Awaitable[bool]] | None = None*, ***, *event_type: None = None*, *adapter_type: None = None*, *max_try_times: int | None = None*, *timeout: int | float | None = None*) → *Event*[Any]

async get(*func: Callable[[EventT], bool | Awaitable[bool]] | None = None*, ***, *event_type: None = None*, *adapter_type: Type[*Adapter*[EventT, Any]]*, *max_try_times: int | None = None*, *timeout: int | float | None = None*) → EventT

async get(*func: Callable[[EventT], bool | Awaitable[bool]] | None = None*, ***, *event_type: Type[EventT]*, *adapter_type: Type[AdapterT] | None = None*, *max_try_times: int | None = None*, *timeout: int | float | None = None*) → EventT

Parameters

- **func** – *True* *None*
- **event_type** – func *None*
- **adapter_type** – func *None*
- **max_try_times** –
- **timeout** –

Returns
func

Raises
GetEventTimeout –

get_adapter(*adapter: str*) → *Adapter*[Any, Any]

get_adapter(*adapter: Type[AdapterT]*) → AdapterT

Parameters
adapter –

Returns

Raises
LookupError –

get_plugin(*name: str*) → Type[*Plugin*[Any, Any, Any]]

Parameters
name –

Returns

Raises
LookupError –

global_state: Dict[Any, Any]

async handle_event(*current_event: Event[Any]*, *, *handle_get: bool = True*, *show_log: bool = True*) → None

stop skip

Parameters

- **current_event** – *Event*
- **handle_get** – get *True*
- **show_log** – *True*

load_adapters(**adapters: Type[Adapter[Any, Any]] | str*) → None

Parameters

***adapters** – *Type[Adapter] str Type[Adapter] str* Python *import path.of.adapter*

load_plugins(**plugins: Type[Plugin[Any, Any, Any]] | str | Path*) → None

Parameters

***plugins** – *Type[Plugin], str pathlib.Path Type[Plugin] str* Python *import path.of.plugin*

pathlib.Path
pathlib.Path("path/of/plugin")

load_plugins_from_dirs(**dirs: Path*) → None

–

Parameters

***dirs** – *pathlib.Path("path/of/plugins/")*

plugin_state: Dict[str, Any]

property plugins: List[Type[*Plugin*[Any, Any, Any]]]

plugins_priority_dict: Dict[int, List[Type[*Plugin*[Any, Any, Any]]]]

reload_plugins() → None

restart() → None

iamai

run() → None

iamai

should_exit: Event

class iamai.**ConfigModel**(***extra_data: Any*)

Bases: BaseModel

iamai

__config_name__

Type

str

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

iamai.**Depends**(*dependency: Type[_T | AsyncContextManager[_T] | ContextManager[_T]] | Callable[[], AsyncGenerator[_T, None]] | Callable[[], Generator[_T, None, None]] | None = None*, *, *use_cache: bool = True*) → _T

Parameters

- **dependency** –
- **use_cache** – *True*

Returns

class iamai.**Event**(*, *adapter: Any*, *type: str | None*, ***extra_data: Any*)

Bases: ABC, BaseModel, Generic[AdapterT]

adapter

Type
Any

type

Type
str | None

__handled__

Type
bool

adapter: Any

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

type: str | None

class iamai.**MessageEvent**(*, *adapter: Any*, *type: str | None*, ***extra_data: Any*)

Bases: *Event*, Generic[AdapterT]

async **ask**(*message: str*, *max_try_times: int | None = None*, *timeout: int | float | None = None*) → Self

reply() get()

Parameters

- **message** –
- **max_try_times** –
- **timeout** –

Returns

async **get**(*, *max_try_times: int | None = None*, *timeout: int | float | None = None*) → Self

Bot get()

Parameters

- **max_try_times** –
- **timeout** –

Returns

Raises

GetEventTimeout –

abstract **get_plain_text**() → str

Returns

abstract async **is_same_sender**(*other: Self*) → bool

Parameters

other –

Returns

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow'}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'adapter': FieldInfo(annotation=Any, required=True), 'type': FieldInfo(annotation=Union[str, NoneType], required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

abstract async reply(*message: str*) → Any

Parameters
message –

Returns

class iamai.**Plugin**

Bases: ABC, Generic[EventT, StateT, ConfigT]

iamai

event

priority

0

Type
ClassVar[int]

block

True

Type
ClassVar[bool]

__plugin_load_type__

iamai

Type
ClassVar[*iamai.plugin.PluginLoadType*]

__plugin_file_path__

PluginLoadType.CLASS *None* Python

Type
ClassVar[str | None]

Config: Type[ConfigT]

block: ClassVar[bool] = False

property bot: *Bot*

property config: ConfigT

event: EventT = InnerDepends(Event)

abstract async handle() → None

rule() *True* iamai

property name: str

priority: ClassVar[int] = 0

abstract async rule() → bool

True

handle()

final skip() → NoReturn

property state: StateT

final stop() → NoReturn

Contributing

Changelog All notable changes to this project will be documented in this file.

The format is based on [Keep a Changelog](https://keepachangelog.com/en/1.0.0/), and this project adheres to [Semantic Versioning](https://semver.org/spec/v2.0.0.html).

[v0.0.3rc3] - 2024-02-23 ### BREAKING CHANGES - due to [*a09bf3f*](https://github.com/retrofor/iamai/commit/a09bf3f8a25693c0f28f1e332c4391529fa69f16) - rewrite kook, apscheduler adapter *(commit by [@HsiangNianian](https://github.com/HsiangNianian))*:

rewrite kook, apscheduler adapter

Refactors - [*a09bf3f*](https://github.com/retrofor/iamai/commit/a09bf3f8a25693c0f28f1e332c4391529fa69f16) - **adapter**: rewrite kook, apscheduler adapter *(commit by [@HsiangNianian](https://github.com/HsiangNianian))*

Chores - [*e3f9845*](https://github.com/retrofor/iamai/commit/e3f98452af7178d767cd72335fe4a6febd0cd042) - **project**: bump version to 0.0.3rc3 *(commit by [@HsiangNianian](https://github.com/HsiangNianian))*

[v0.0.3b5] - 2024-02-23 ### Bug Fixes - [*76fa3e0*](https://github.com/retrofor/iamai/commit/76fa3e0cf3f1120df68c817c0a3e2cb9f23e631e) - **workflow**: rename "onebot11" to "cqhttp" *(commit by [@HsiangNianian](https://github.com/HsiangNianian))*

Chores - [*4fe3918*](https://github.com/retrofor/iamai/commit/4fe39182a361eb60b559ee57a3a956e0757a1128) - **readme**: upate *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*bae16f4*](https://github.com/retrofor/iamai/commit/bae16f4025bfaa0c8296924865f232c0234699c5) - **readme**: update *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*281cf90*](https://github.com/retrofor/iamai/commit/281cf904379797b9fec74e9621d92fb751fce88a) - **project**: update readme.text *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*663a70f*](https://github.com/retrofor/iamai/commit/663a70ffb1bbdaaebc425629f53d49f073d2cf26) - **project**: bump version into 0.0.3b4 *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*6565229*](https://github.com/retrofor/iamai/commit/65652292cf7de7f772e19f82eb502a68c986df28) - **project**: bump version to 0.0.3b5 *(commit by [@HsiangNianian](https://github.com/HsiangNianian))*

[v0.0.3b3] - 2024-02-22 ### BREAKING CHANGES - due to [*4e8a96f*](https://github.com/retrofor/iamai/commit/4e8a96f25665c08314d5e68e338327fa3f65e25a) - rename adapter "onebot11" -> "cqhttp" *(commit by [@HsiangNianian](https://github.com/HsiangNianian))*:

rename adapter "onebot11" -> "cqhttp"

Bug Fixes - [*23d0894*](https://github.com/retrofor/iamai/commit/23d0894c3edba5c88c55082a0dd345267a266c8a) - **readme**: format rst *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*22b40d0*](https://github.com/retrofor/iamai/commit/22b40d0af7e0b0255411f9cef749c3dbe95a3c7f) - **workflow**: build api for all packages *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*e366a34*](https://github.com/retrofor/iamai/commit/e366a34f270890e260c12eee4b96a9e85defd5f6) - remove *readme* key and add *long_description_content*. *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*5a1c0e2*](https://github.com/retrofor/iamai/commit/5a1c0e21094857bf8f997823b8067d27887d11e2) - **workflow**: resolve cp -r not specified error *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*0309a33*](https://github.com/retrofor/iamai/commit/0309a33d20634e00828d081dba53ff01a83bab45) - **workflow**: rename dir name *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*d191c0c*](https://github.com/retrofor/iamai/commit/d191c0ccbbebfe90d78659f05b3699a7562b3a49) - **project**: update url and readme file *(commit by [@HsiangNianian](https://github.com/HsiangNianian))*

Refactors - [*4e8a96f*](https://github.com/retrofor/iamai/commit/4e8a96f25665c08314d5e68e338327fa3f65e25a) - **adapter**: rename adapter "onebot11" -> "cqhttp" *(commit by [@HsiangNianian](https://github.com/HsiangNianian))*

Chores - [*283d1af*](https://github.com/retrofor/iamai/commit/283d1af748a1c309bd6c38b1aabf820e9b56d1e3) - **docs**: update api docs with sphinx-apidoc *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*c7bda67*](https://github.com/retrofor/iamai/commit/c7bda6707669d21141f0af88a7dadfffa4b3783e) - **workflow**: update api build step *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*a69aff7*](https://github.com/retrofor/iamai/commit/a69aff74c1d3f967f9825910d2de13f2243517c7) - **docs**: update conf.py *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*b024674*](https://github.com/retrofor/iamai/commit/b0246747ddc95eced7a68a9f04f22ac2a8e6cf7f) - **docs**: change html title *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*607af84*](https://github.com/retrofor/iamai/commit/607af845fa41c4c9c9612512092494c399a896be) - optimize all comments *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*aa14246*](https://github.com/retrofor/iamai/commit/aa14246e0c44fe752d0e372bdbd9654008db45f3) - **docs**: update api docs with sphinx-apidoc *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*22bfbed*](https://github.com/retrofor/iamai/commit/22bfbed1dc21384139ad527e8c689692a73e419f) - **workflow**: delete api dir *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*1216d89*](https://github.com/retrofor/iamai/commit/1216d89493fd259f4c3df8c9b55c1b956f17acf5) - **docs**: format changelog *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*a67637f*](https://github.com/retrofor/iamai/commit/a67637f2936aa0c19ae3b9f0a3b6712b70c9decd) - **docs**: fix file name *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*7172a20*](https://github.com/retrofor/iamai/commit/7172a204c218ec7da69a6504b6ac20c792b94f6a) - **adapter**: apscheduler adpater update deps *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*6d4ef4a*](https://github.com/retrofor/iamai/commit/6d4ef4a60e98ba66528e629eb0f9a76e072d0d24) - **packages**: update LICENSE *(commit by [@HsiangNianian](https://github.com/HsiangNianian))*

[v0.0.3b2] - 2024-02-14 ### Bug Fixes - [*b8e1178*](https://github.com/retrofor/iamai/commit/b8e11784375b670293d4c4cc9f455a2c1c3a93dd) - change project readme file suffix *(commit by [@HsiangNianian](https://github.com/HsiangNianian))*

Chores - [*83adf17*](https://github.com/retrofor/iamai/commit/83adf171f47a7927c2f156d0c2aa37a2aca40f50) - bump version to 003b2 *(commit by [@HsiangNianian](https://github.com/HsiangNianian))*

[v0.0.3b1] - 2024-01-29 ### Refactors - [*1fa6516*](https://github.com/retrofor/iamai/commit/1fa6516950fc2c1a967154ea916c580efb352d39) - rename example to examples *(commit by [@HsiangNianian](https://github.com/HsiangNianian))*

Chores - [*5d7f813*](https://github.com/retrofor/iamai/commit/5d7f81398bee5dd8926580e4f41baab35d0b38a4) - **docs**: update api docs *(PR [#273](https://github.com/retrofor/iamai/pull/273) by [@HsiangNianian](https://github.com/HsiangNianian))* - [*fe9b8cb*](https://github.com/retrofor/iamai/commit/fe9b8cbee74b1d470c6033550f99a9f6afbb8d2a) - **workflows**: disable useGitmojis *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*f5e26df*](https://github.com/retrofor/iamai/commit/f5e26df94e10254e55b9de7f632c069c10cc1433) - **workflows**: update api build steps *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*f222434*](https://github.com/retrofor/iamai/commit/f222434831b73f6a8a940498b670b45f33b96995) - **deps**: bump ncipollo/release-action from 1.12.0 to 1.13.0 *(PR [#275](https://github.com/retrofor/iamai/pull/275) by [@dependabot[bot]](https://github.com/apps/dependabot))* - [*aa61cf1*](https://github.com/retrofor/iamai/commit/aa61cf10d711c33c14ae22c641a96c6deec19570) - **deps**: bump stefanzweifel/git-auto-commit-action from 4 to 5 *(PR [#274](https://github.com/retrofor/iamai/pull/274) by [@dependabot[bot]](https://github.com/apps/dependabot))*

[v0.0.3a3] - 2024-01-28 ### :wrench: Chores - [*84be7fd*](https://github.com/retrofor/iamai/commit/84be7fdae9f828e9422d209a1919a6bdbec29ac2) - **workflows**: remove step copy credits *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*9732bf1*](https://github.com/retrofor/iamai/commit/9732bf1c8bc617e324c0b1f1b2500caf78ba79b1) - **version**: bump version from 0.0.3a2 to 0.0.3a3 *(commit by [@HsiangNianian](https://github.com/HsiangNianian))*

[v0.0.3a2] - 2024-01-28 ### :boom: BREAKING CHANGES - due to [*7f1ceea*](https://github.com/retrofor/iamai/commit/7f1ceea0763e6914a6d52b398ee9959f98470b52) - cancel building the credit with workflow *(commit by [@HsiangNianian](https://github.com/HsiangNianian))*:

cancel building the credit with workflow

:bug: Bug Fixes - [*ea0ada8*](https://github.com/retrofor/iamai/commit/ea0ada86ac23f805cd76b3e45d6971fed638468a) - **workflow**: rename changelog.md to CHANGELOG.md and replace the path *(commit by [@HsiangNianian](https://github.com/HsiangNianian))*

:recycle: Refactors - [*59adea4*](https://github.com/retrofor/iamai/commit/59adea4e43ac2c2a20e44b7622ee9c05670b728b) - **example**: rename tests dir to examples dir *(commit by [@HsiangNianian](https://github.com/HsiangNianian))*

:wrench: Chores - [*0428f79*](https://github.com/retrofor/iamai/commit/0428f79806188399cdd09f9cc459841228429923) - **workflow**: change regex pattern *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*4651ea8*](https://github.com/retrofor/iamai/commit/4651ea8a81a449e4398a641664bb831bd6b7b518) - **workflow**: revoke changelog.md *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*a15ab94*](https://github.com/retrofor/iamai/commit/a15ab948c12ae5be9dc6d2133ec85b793927d6a4) - **deps**: remove sophia-doc *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*7f1ceea*](https://github.com/retrofor/iamai/commit/7f1ceea0763e6914a6d52b398ee9959f98470b52) - **credits**: cancel building the credit with workflow *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*6048e01*](https://github.com/retrofor/iamai/commit/6048e01a34ba795e2b98d46bb9a9d0421dcad377) - **workflows**: delete relese-drafter.yml *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*bba1a90*](https://github.com/retrofor/iamai/commit/bba1a902f63b8ce3208f588e19dd1f95b0d8e578) - update ignored files *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*4b3fadf*](https://github.com/retrofor/iamai/commit/4b3fadfcb09c8f6c41ab794e10e45ece535c9a98) - delete example/config.toml *(commit by [@HsiangNianian](https://github.com/HsiangNianian))* - [*b3679ba*](https://github.com/retrofor/iamai/commit/b3679ba5beabe4ea913dda4507b5e795dcc39c4d) - **version**: bump version 0.0.3a1 to 0.0.3a2 *(commit by [@HsiangNianian](https://github.com/HsiangNianian))*

[v0.0.3a2]: https://github.com/retrofor/iamai/compare/v0.0.3a1...v0.0.3a2 [v0.0.3a3]: https://github.com/retrofor/iamai/compare/v0.0.3a2...v0.0.3a3 [v0.0.3b1]: https://github.com/retrofor/iamai/compare/v0.0.3a3...v0.0.3
[v0.0.3b2]: https://github.com/retrofor/iamai/compare/v0.0.3b1...v0.0.3b2 [v0.0.3b3]: https://github.com/retrofor/iamai/compare/v0.0.3b2...v0.0.3b3 [v0.0.3b5]: https://github.com/retrofor/iamai/compare/v0.0.3b3...v0.0.3
[v0.0.3rc3]: https://github.com/retrofor/iamai/compare/v0.0.3rc1...v0.0.3rc3

CHAPTER

THIRTEEN

COPYING

CHAPTER

FOURTEEN

DEPENDENCIES

14.1 aiohttp (3.9.1)

14.1.1 Declared Licenses

Apache-2.0

```
  Copyright aio-libs contributors.

  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
```

14.1.2 Other Licenses

MIT

```
Copyright (c)  Fedor Indutny, 2018.
Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
```

(continues on next page)

(continued from previous page)

```
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

14.2 aiosignal (1.3.1)

14.2.1 Declared Licenses

Apache-2.0

```
Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

   TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

   1. Definitions.

      "License" shall mean the terms and conditions for use, reproduction,
      and distribution as defined by Sections 1 through 9 of this document.

      "Licensor" shall mean the copyright owner or entity authorized by
      the copyright owner that is granting the License.

      "Legal Entity" shall mean the union of the acting entity and all
      other entities that control, are controlled by, or are under common
      control with that entity. For the purposes of this definition,
      "control" means (i) the power, direct or indirect, to cause the
      direction or management of such entity, whether by contract or
      otherwise, or (ii) ownership of fifty percent (50%) or more of the
      outstanding shares, or (iii) beneficial ownership of such entity.

      "You" (or "Your") shall mean an individual or Legal Entity
      exercising permissions granted by this License.

      "Source" form shall mean the preferred form for making modifications,
      including but not limited to software source code, documentation
      source, and configuration files.

      "Object" form shall mean any form resulting from mechanical
      transformation or translation of a Source form, including but
      not limited to compiled object code, generated documentation,
      and conversions to other media types.

      "Work" shall mean the work of authorship, whether in Source or
      Object form, made available under the License, as indicated by a
```

(continues on next page)

(continued from previous page)

```
    copyright notice that is included in or attached to the work
    (an example is provided in the Appendix below).

    "Derivative Works" shall mean any work, whether in Source or Object
    form, that is based on (or derived from) the Work and for which the
    editorial revisions, annotations, elaborations, or other modifications
    represent, as a whole, an original work of authorship. For the purposes
    of this License, Derivative Works shall not include works that remain
    separable from, or merely link (or bind by name) to the interfaces of,
    the Work and Derivative Works thereof.

    "Contribution" shall mean any work of authorship, including
    the original version of the Work and any modifications or additions
    to that Work or Derivative Works thereof, that is intentionally
    submitted to Licensor for inclusion in the Work by the copyright owner
    or by an individual or Legal Entity authorized to submit on behalf of
    the copyright owner. For the purposes of this definition, "submitted"
    means any form of electronic, verbal, or written communication sent
    to the Licensor or its representatives, including but not limited to
    communication on electronic mailing lists, source code control systems,
    and issue tracking systems that are managed by, or on behalf of, the
    Licensor for the purpose of discussing and improving the Work, but
    excluding communication that is conspicuously marked or otherwise
    designated in writing by the copyright owner as "Not a Contribution."

    "Contributor" shall mean Licensor and any individual or Legal Entity
    on behalf of whom a Contribution has been received by Licensor and
    subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
    this License, each Contributor hereby grants to You a perpetual,
    worldwide, non-exclusive, no-charge, royalty-free, irrevocable
    copyright license to reproduce, prepare Derivative Works of,
    publicly display, publicly perform, sublicense, and distribute the
    Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
    this License, each Contributor hereby grants to You a perpetual,
    worldwide, non-exclusive, no-charge, royalty-free, irrevocable
    (except as stated in this section) patent license to make, have made,
    use, offer to sell, sell, import, and otherwise transfer the Work,
    where such license applies only to those patent claims licensable
    by such Contributor that are necessarily infringed by their
    Contribution(s) alone or by combination of their Contribution(s)
    with the Work to which such Contribution(s) was submitted. If You
    institute patent litigation against any entity (including a
    cross-claim or counterclaim in a lawsuit) alleging that the Work
    or a Contribution incorporated within the Work constitutes direct
    or contributory patent infringement, then any patent licenses
    granted to You under this License for that Work shall terminate
    as of the date such litigation is filed.
```

(continues on next page)

(continued from previous page)

```
   4. Redistribution. You may reproduce and distribute copies of the
      Work or Derivative Works thereof in any medium, with or without
      modifications, and in Source or Object form, provided that You
      meet the following conditions:

      (a) You must give any other recipients of the Work or
          Derivative Works a copy of this License; and

      (b) You must cause any modified files to carry prominent notices
          stating that You changed the files; and

      (c) You must retain, in the Source form of any Derivative Works
          that You distribute, all copyright, patent, trademark, and
          attribution notices from the Source form of the Work,
          excluding those notices that do not pertain to any part of
          the Derivative Works; and

      (d) If the Work includes a "NOTICE" text file as part of its
          distribution, then any Derivative Works that You distribute must
          include a readable copy of the attribution notices contained
          within such NOTICE file, excluding those notices that do not
          pertain to any part of the Derivative Works, in at least one
          of the following places: within a NOTICE text file distributed
          as part of the Derivative Works; within the Source form or
          documentation, if provided along with the Derivative Works; or,
          within a display generated by the Derivative Works, if and
          wherever such third-party notices normally appear. The contents
          of the NOTICE file are for informational purposes only and
          do not modify the License. You may add Your own attribution
          notices within Derivative Works that You distribute, alongside
          or as an addendum to the NOTICE text from the Work, provided
          that such additional attribution notices cannot be construed
          as modifying the License.

      You may add Your own copyright statement to Your modifications and
      may provide additional or different license terms and conditions
      for use, reproduction, or distribution of Your modifications, or
      for any such Derivative Works as a whole, provided Your use,
      reproduction, and distribution of the Work otherwise complies with
      the conditions stated in this License.

   5. Submission of Contributions. Unless You explicitly state otherwise,
      any Contribution intentionally submitted for inclusion in the Work
      by You to the Licensor shall be under the terms and conditions of
      this License, without any additional terms or conditions.
      Notwithstanding the above, nothing herein shall supersede or modify
      the terms of any separate license agreement you may have executed
      with Licensor regarding such Contributions.

   6. Trademarks. This License does not grant permission to use the trade
      names, trademarks, service marks, or product names of the Licensor,
      except as required for reasonable and customary use in describing the
```

(continues on next page)

(continued from previous page)

```
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets "{}"
   replaced with your own identifying information. (Don't include
   the brackets!)  The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
   same "printed page" as the copyright notice for easier
   identification within third-party archives.

Copyright 2013-2019 Nikolay Kim and Andrew Svetlov

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
```

(continues on next page)

(continued from previous page)

```
   You may obtain a copy of the License at

       http://www.apache.org/licenses/LICENSE-2.0

   Unless required by applicable law or agreed to in writing, software
   distributed under the License is distributed on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   See the License for the specific language governing permissions and
   limitations under the License.
```

14.2.2 Other Licenses

14.3 annotated-types (0.6.0)

14.3.1 Declared Licenses

MIT

```
The MIT License (MIT)

Copyright (c) 2022 the contributors

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

14.3.2 Other Licenses

14.4 anyio (4.0.0)

14.4.1 Declared Licenses

MIT

```
The MIT License (MIT)

Copyright (c) 2018 Alex Grönholm

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

14.4.2 Other Licenses

14.5 async-timeout (4.0.3)

14.5.1 Declared Licenses

Apache-2.0

```
Copyright 2016-2020 aio-libs collaboration.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

   http://www.apache.org/licenses/LICENSE-2.0
```

(continues on next page)

(continued from previous page)

```
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

14.5.2 Other Licenses

14.6 attrs (23.1.0)

14.6.1 Declared Licenses

MIT

```
The MIT License (MIT)

Copyright (c) 2015 Hynek Schlawack and the attrs contributors

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

14.6.2 Other Licenses

14.7 certifi (2023.11.17)

14.7.1 Declared Licenses

MPL-2.0

```
This package contains a modified version of ca-bundle.crt:

ca-bundle.crt -- Bundle of CA Root Certificates

This is a bundle of X.509 certificates of public Certificate Authorities
(CA). These were automatically extracted from Mozilla's root certificates
file (certdata.txt).  This file can be found in the mozilla source tree:
https://hg.mozilla.org/mozilla-central/file/tip/security/nss/lib/ckfw/builtins/certdata.
↪txt
It contains the certificates in PEM format and therefore
can be directly used with curl / libcurl / php_curl, or with
an Apache+mod_ssl webserver for SSL client authentication.
Just configure this file as the SSLCACertificateFile.#

***** BEGIN LICENSE BLOCK *****
This Source Code Form is subject to the terms of the Mozilla Public License,
v. 2.0. If a copy of the MPL was not distributed with this file, You can obtain
one at http://mozilla.org/MPL/2.0/.

***** END LICENSE BLOCK *****
@(#) $RCSfile: certdata.txt,v $ $Revision: 1.80 $ $Date: 2011/11/03 15:11:58 $
```

14.7.2 Other Licenses

14.8 charset-normalizer (3.3.2)

14.8.1 Declared Licenses

MIT

```
MIT License

Copyright (c) 2019 TAHRI Ahmed R.

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
```

(continues on next page)

(continued from previous page)

```
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

14.8.2 Other Licenses

14.9 click (8.1.7)

14.9.1 Declared Licenses

BSD-3-Clause

```
Copyright 2014 Pallets

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1\.  Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.

2\.  Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.

3\.  Neither the name of the copyright holder nor the names of its
    contributors may be used to endorse or promote products derived from
    this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

14.9.2 Other Licenses

14.10 colorama (0.4.6)

14.10.1 Declared Licenses

BSD-3-Clause

```
Copyright (c) 2010 Jonathan Hartley
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

\* Redistributions of source code must retain the above copyright notice, this
  list of conditions and the following disclaimer.

\* Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.

\* Neither the name of the copyright holders, nor those of its contributors
  may be used to endorse or promote products derived from this software without
  specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

14.10.2 Other Licenses

14.11 exceptiongroup (1.1.3)

14.11.1 Declared Licenses

Python-2.0, MIT

```
1\. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the␣
↪Individual or Organization ("Licensee") accessing and otherwise using this software (
↪"Python") in source or binary form and its associated documentation.
   2. Subject to the terms and conditions of this License Agreement, PSF hereby grants␣
↪Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test,␣
↪perform and/or display publicly, prepare derivative works, distribute, and otherwise␣
↪use Python alone or in any derivative version, provided, however, that PSF's License␣
↪Agreement and PSF's notice of copyright, i.e., "Copyright (c) 2001, 2002, 2003, 2004,␣
↪2005, 2006 Python Software Foundation; All Rights Reserved" are retained in Python␣
↪alone or in any derivative version prepared by Licensee.
   3. In the event Licensee prepares a derivative work that is based on or incorporates␣
↪Python or any part thereof, and wants to make the derivative work available to others␣
↪as provided herein, then Licensee hereby agrees to include in any such work a brief␣
↪summary of the changes made to Python.
   4. PSF is making Python available to Licensee on an "AS IS" basis. PSF MAKES NO␣
↪REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT␣
↪LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF␣
↪MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON WILL␣
↪NOT INFRINGE ANY THIRD PARTY RIGHTS.
   5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON FOR ANY␣
↪INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING,␣
↪DISTRIBUTING, OR OTHERWISE USING PYTHON, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF␣
↪THE POSSIBILITY THEREOF.
   6. This License Agreement will automatically terminate upon a material breach of its␣
↪terms and conditions.
   7. Nothing in this License Agreement shall be deemed to create any relationship of␣
↪agency, partnership, or joint venture between PSF and Licensee. This License Agreement␣
↪does not grant permission to use PSF trademarks or trade name in a trademark sense to␣
↪endorse or promote products or services of Licensee, or any third party.
   8. By copying, installing or otherwise using Python, Licensee agrees to be bound by␣
↪the terms and conditions of this License Agreement.
   1. This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an office at 160␣
↪Saratoga Avenue, Santa Clara, CA 95051, and the Individual or Organization ("Licensee
↪") accessing and otherwise using this software in source or binary form and its␣
↪associated documentation ("the Software").
   2. Subject to the terms and conditions of this BeOpen Python License Agreement,␣
↪BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license to␣
↪reproduce, analyze, test, perform and/or display publicly, prepare derivative works,␣
↪distribute, and otherwise use the Software alone or in any derivative version,␣
↪provided, however, that the BeOpen Python License is retained in the Software, alone␣
↪or in any derivative version prepared by Licensee.
   3. BeOpen is making the Software available to Licensee on an "AS IS" basis. BEOPEN␣
↪MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT␣
↪LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF␣
↪MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE␣
↪WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
   4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY␣
↪INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING, MODIFYING␣
↪OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE␣
↪POSSIBILITY THEREOF.
   5. This License Agreement will automatically terminate upon a material breach of its␣
↪terms and conditions.
   6. This License Agreement shall be governed by and interpreted in all respects by the␣
```

(continues on next page)

(continued from previous page)

```
↪law of the State of California, excluding conflict of law provisions. Nothing in this␣
↪License Agreement shall be deemed to create any relationship of agency, partnership,␣
↪or joint venture between BeOpen and Licensee. This License Agreement does not grant␣
↪permission to use BeOpen trademarks or trade names in a trademark sense to endorse or␣
↪promote products or services of Licensee, or any third party. As an exception, the
↪"BeOpen Python" logos available at http://www.pythonlabs.com/logos.html may be used␣
↪according to the permissions granted on that web page.
   7. By copying, installing or otherwise using the software, Licensee agrees to be␣
↪bound by the terms and conditions of this License Agreement.
   1. This LICENSE AGREEMENT is between the Corporation for National Research␣
↪Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191 ("CNRI"),␣
↪and the Individual or Organization ("Licensee") accessing and otherwise using Python 1.
↪6, beta 1 software in source or binary form and its associated documentation, as␣
↪released at the www.python.org Internet site on August 4, 2000 ("Python 1.6b1").
   2. Subject to the terms and conditions of this License Agreement, CNRI hereby grants␣
↪Licensee a non-exclusive, royalty-free, world-wide license to reproduce, analyze, test,
↪ perform and/or display publicly, prepare derivative works, distribute, and otherwise␣
↪use Python 1.6b1 alone or in any derivative version, provided, however, that CNRIs␣
↪License Agreement is retained in Python 1.6b1, alone or in any derivative version␣
↪prepared by Licensee.
   Alternately, in lieu of CNRIs License Agreement, Licensee may substitute the␣
↪following text (omitting the quotes): "Python 1.6, beta 1, is made available subject␣
↪to the terms and conditions in CNRIs License Agreement. This Agreement may be located␣
↪on the Internet using the following unique, persistent identifier (known as a handle):␣
↪1895.22/1011. This Agreement may also be obtained from a proxy server on the Internet␣
↪using the URL:http://hdl.handle.net/1895.22/1011".
   3. In the event Licensee prepares a derivative work that is based on or incorporates␣
↪Python 1.6b1 or any part thereof, and wants to make the derivative work available to␣
↪the public as provided herein, then Licensee hereby agrees to indicate in any such␣
↪work the nature of the modifications made to Python 1.6b1.
   4. CNRI is making Python 1.6b1 available to Licensee on an "AS IS" basis. CNRI MAKES␣
↪NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT␣
↪LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF␣
↪MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6b1␣
↪WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
   5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY␣
↪INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING, MODIFYING␣
↪OR DISTRIBUTING PYTHON 1.6b1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE␣
↪POSSIBILITY THEREOF.
   6. This License Agreement will automatically terminate upon a material breach of its␣
↪terms and conditions.
   7. This License Agreement shall be governed by and interpreted in all respects by the␣
↪law of the State of Virginia, excluding conflict of law provisions. Nothing in this␣
↪License Agreement shall be deemed to create any relationship of agency, partnership,␣
↪or joint venture between CNRI and Licensee. This License Agreement does not grant␣
↪permission to use CNRI trademarks or trade name in a trademark sense to endorse or␣
↪promote products or services of Licensee, or any third party.
   8. By clicking on the "ACCEPT" button where indicated, or by copying, installing or␣
↪otherwise using Python 1.6b1, Licensee agrees to be bound by the terms and conditions␣
↪of this License Agreement.
Copyright (c) 1991 - 1995, Stichting Mathematisch Centrum Amsterdam, The Netherlands.␣
↪All rights reserved.
```

(continues on next page)

(continued from previous page)

```
Permission to use, copy, modify, and distribute this software and its documentation for␣
↪any purpose and without fee is hereby granted, provided that the above copyright␣
↪notice appear in all copies and that both that copyright notice and this permission␣
↪notice appear in supporting documentation, and that the name of Stichting Mathematisch␣
↪Centrum or CWI not be used in advertising or publicity pertaining to distribution of␣
↪the software without specific, written prior permission.
STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,␣
↪INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL␣
↪STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL␣
↪DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER␣
↪IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN␣
↪CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
```

```
The MIT License (MIT)

Copyright (c) 2022 Alex Grönholm

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.


This project contains code copied from the Python standard library.
The following is the required license notice for those parts.

PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2
--------------------------------------------

1\. This LICENSE AGREEMENT is between the Python Software Foundation
("PSF"), and the Individual or Organization ("Licensee") accessing and
otherwise using this software ("Python") in source or binary form and
its associated documentation.

2\. Subject to the terms and conditions of this License Agreement, PSF hereby
grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce,
analyze, test, perform and/or display publicly, prepare derivative works,
distribute, and otherwise use Python alone or in any derivative version,
provided, however, that PSF's License Agreement and PSF's notice of copyright,
i.e., "Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010,
```

(continues on next page)

(continued from previous page)

```
2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022 Python Software␣
↪Foundation;
All Rights Reserved" are retained in Python alone or in any derivative version
prepared by Licensee.

3\. In the event Licensee prepares a derivative work that is based on
or incorporates Python or any part thereof, and wants to make
the derivative work available to others as provided herein, then
Licensee hereby agrees to include in any such work a brief summary of
the changes made to Python.

4\. PSF is making Python available to Licensee on an "AS IS"
basis.  PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR
IMPLIED.  BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND
DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS
FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON WILL NOT
INFRINGE ANY THIRD PARTY RIGHTS.

5\. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON
FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS
A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON,
OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6\. This License Agreement will automatically terminate upon a material
breach of its terms and conditions.

7\. Nothing in this License Agreement shall be deemed to create any
relationship of agency, partnership, or joint venture between PSF and
Licensee.  This License Agreement does not grant permission to use PSF
trademarks or trade name in a trademark sense to endorse or promote
products or services of Licensee, or any third party.

8\. By copying, installing or otherwise using Python, Licensee
agrees to be bound by the terms and conditions of this License
Agreement.
```

14.11.2 Other Licenses

14.12 frozenlist (1.4.0)

14.12.1 Declared Licenses

Apache-2.0

```
Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/
```

(continues on next page)

(continued from previous page)

```
TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by
   the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all
   other entities that control, are controlled by, or are under common
   control with that entity. For the purposes of this definition,
   "control" means (i) the power, direct or indirect, to cause the
   direction or management of such entity, whether by contract or
   otherwise, or (ii) ownership of fifty percent (50%) or more of the
   outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity
   exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications,
   including but not limited to software source code, documentation
   source, and configuration files.

   "Object" form shall mean any form resulting from mechanical
   transformation or translation of a Source form, including but
   not limited to compiled object code, generated documentation,
   and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or
   Object form, made available under the License, as indicated by a
   copyright notice that is included in or attached to the work
   (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object
   form, that is based on (or derived from) the Work and for which the
   editorial revisions, annotations, elaborations, or other modifications
   represent, as a whole, an original work of authorship. For the purposes
   of this License, Derivative Works shall not include works that remain
   separable from, or merely link (or bind by name) to the interfaces of,
   the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including
   the original version of the Work and any modifications or additions
   to that Work or Derivative Works thereof, that is intentionally
   submitted to Licensor for inclusion in the Work by the copyright owner
   or by an individual or Legal Entity authorized to submit on behalf of
   the copyright owner. For the purposes of this definition, "submitted"
   means any form of electronic, verbal, or written communication sent
   to the Licensor or its representatives, including but not limited to
```

(continues on next page)

(continued from previous page)

```
    communication on electronic mailing lists, source code control systems,
    and issue tracking systems that are managed by, or on behalf of, the
    Licensor for the purpose of discussing and improving the Work, but
    excluding communication that is conspicuously marked or otherwise
    designated in writing by the copyright owner as "Not a Contribution."

    "Contributor" shall mean Licensor and any individual or Legal Entity
    on behalf of whom a Contribution has been received by Licensor and
    subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
    this License, each Contributor hereby grants to You a perpetual,
    worldwide, non-exclusive, no-charge, royalty-free, irrevocable
    copyright license to reproduce, prepare Derivative Works of,
    publicly display, publicly perform, sublicense, and distribute the
    Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
    this License, each Contributor hereby grants to You a perpetual,
    worldwide, non-exclusive, no-charge, royalty-free, irrevocable
    (except as stated in this section) patent license to make, have made,
    use, offer to sell, sell, import, and otherwise transfer the Work,
    where such license applies only to those patent claims licensable
    by such Contributor that are necessarily infringed by their
    Contribution(s) alone or by combination of their Contribution(s)
    with the Work to which such Contribution(s) was submitted. If You
    institute patent litigation against any entity (including a
    cross-claim or counterclaim in a lawsuit) alleging that the Work
    or a Contribution incorporated within the Work constitutes direct
    or contributory patent infringement, then any patent licenses
    granted to You under this License for that Work shall terminate
    as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
    Work or Derivative Works thereof in any medium, with or without
    modifications, and in Source or Object form, provided that You
    meet the following conditions:

    (a) You must give any other recipients of the Work or
        Derivative Works a copy of this License; and

    (b) You must cause any modified files to carry prominent notices
        stating that You changed the files; and

    (c) You must retain, in the Source form of any Derivative Works
        that You distribute, all copyright, patent, trademark, and
        attribution notices from the Source form of the Work,
        excluding those notices that do not pertain to any part of
        the Derivative Works; and

    (d) If the Work includes a "NOTICE" text file as part of its
        distribution, then any Derivative Works that You distribute must
```

(continues on next page)

(continued from previous page)

```
        include a readable copy of the attribution notices contained
        within such NOTICE file, excluding those notices that do not
        pertain to any part of the Derivative Works, in at least one
        of the following places: within a NOTICE text file distributed
        as part of the Derivative Works; within the Source form or
        documentation, if provided along with the Derivative Works; or,
        within a display generated by the Derivative Works, if and
        wherever such third-party notices normally appear. The contents
        of the NOTICE file are for informational purposes only and
        do not modify the License. You may add Your own attribution
        notices within Derivative Works that You distribute, alongside
        or as an addendum to the NOTICE text from the Work, provided
        that such additional attribution notices cannot be construed
        as modifying the License.

    You may add Your own copyright statement to Your modifications and
    may provide additional or different license terms and conditions
    for use, reproduction, or distribution of Your modifications, or
    for any such Derivative Works as a whole, provided Your use,
    reproduction, and distribution of the Work otherwise complies with
    the conditions stated in this License.

  5. Submission of Contributions. Unless You explicitly state otherwise,
    any Contribution intentionally submitted for inclusion in the Work
    by You to the Licensor shall be under the terms and conditions of
    this License, without any additional terms or conditions.
    Notwithstanding the above, nothing herein shall supersede or modify
    the terms of any separate license agreement you may have executed
    with Licensor regarding such Contributions.

  6. Trademarks. This License does not grant permission to use the trade
    names, trademarks, service marks, or product names of the Licensor,
    except as required for reasonable and customary use in describing the
    origin of the Work and reproducing the content of the NOTICE file.

  7. Disclaimer of Warranty. Unless required by applicable law or
    agreed to in writing, Licensor provides the Work (and each
    Contributor provides its Contributions) on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
    implied, including, without limitation, any warranties or conditions
    of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
    PARTICULAR PURPOSE. You are solely responsible for determining the
    appropriateness of using or redistributing the Work and assume any
    risks associated with Your exercise of permissions under this License.

  8. Limitation of Liability. In no event and under no legal theory,
    whether in tort (including negligence), contract, or otherwise,
    unless required by applicable law (such as deliberate and grossly
    negligent acts) or agreed to in writing, shall any Contributor be
    liable to You for damages, including any direct, indirect, special,
    incidental, or consequential damages of any character arising as a
    result of this License or out of the use or inability to use the
```

(continues on next page)

(continued from previous page)

```
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets "{}"
   replaced with your own identifying information. (Don't include
   the brackets!)  The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
   same "printed page" as the copyright notice for easier
   identification within third-party archives.

Copyright 2013-2019 Nikolay Kim and Andrew Svetlov

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

14.12.2 Other Licenses

14.13 iamai (0.0.2)

14.13.1 Declared Licenses

MIT

```
MIT License

Copyright (c) 2023 Retro for wut?

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

14.13.2 Other Licenses

14.14 idna (3.4)

14.14.1 Declared Licenses

BSD-3-Clause

```
BSD 3-Clause License

Copyright (c) 2013-2021, Kim Davies
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:
```

(continues on next page)

(continued from previous page)

```
1\. Redistributions of source code must retain the above copyright notice, this
   list of conditions and the following disclaimer.

2\. Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.

3\. Neither the name of the copyright holder nor the names of its
   contributors may be used to endorse or promote products derived from
   this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

14.14.2 Other Licenses

14.15 loguru (0.7.2)

14.15.1 Declared Licenses

MIT

```
MIT License

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
```

(continues on next page)

(continued from previous page)

```
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

14.15.2 Other Licenses

14.16 multidict (6.0.4)

14.16.1 Declared Licenses

Apache-2.0

```
   Copyright 2016-2021 Andrew Svetlov and aio-libs team

   Licensed under the Apache License, Version 2.0 (the "License");
   you may not use this file except in compliance with the License.
   You may obtain a copy of the License at

       http://www.apache.org/licenses/LICENSE-2.0

   Unless required by applicable law or agreed to in writing, software
   distributed under the License is distributed on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   See the License for the specific language governing permissions and
   limitations under the License.
```

14.16.2 Other Licenses

14.17 pydantic (2.5.0)

14.17.1 Declared Licenses

MIT

```
The MIT License (MIT)

Copyright (c) 2017 to present Pydantic Services Inc. and individual contributors.

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
```

(continues on next page)

(continued from previous page)

```
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

14.17.2 Other Licenses

14.18 pydantic-core (2.14.1)

14.18.1 Declared Licenses

MIT

```
The MIT License (MIT)

Copyright (c) 2022 Samuel Colvin

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

14.18.2 Other Licenses

14.19 requests (2.31.0)

14.19.1 Declared Licenses

Apache-2.0

```
                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

   TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

   1. Definitions.

      "License" shall mean the terms and conditions for use, reproduction,
      and distribution as defined by Sections 1 through 9 of this document.

      "Licensor" shall mean the copyright owner or entity authorized by
      the copyright owner that is granting the License.

      "Legal Entity" shall mean the union of the acting entity and all
      other entities that control, are controlled by, or are under common
      control with that entity. For the purposes of this definition,
      "control" means (i) the power, direct or indirect, to cause the
      direction or management of such entity, whether by contract or
      otherwise, or (ii) ownership of fifty percent (50%) or more of the
      outstanding shares, or (iii) beneficial ownership of such entity.

      "You" (or "Your") shall mean an individual or Legal Entity
      exercising permissions granted by this License.

      "Source" form shall mean the preferred form for making modifications,
      including but not limited to software source code, documentation
      source, and configuration files.

      "Object" form shall mean any form resulting from mechanical
      transformation or translation of a Source form, including but
      not limited to compiled object code, generated documentation,
      and conversions to other media types.

      "Work" shall mean the work of authorship, whether in Source or
      Object form, made available under the License, as indicated by a
      copyright notice that is included in or attached to the work
      (an example is provided in the Appendix below).

      "Derivative Works" shall mean any work, whether in Source or Object
      form, that is based on (or derived from) the Work and for which the
```

(continues on next page)

(continued from previous page)

```
    editorial revisions, annotations, elaborations, or other modifications
    represent, as a whole, an original work of authorship. For the purposes
    of this License, Derivative Works shall not include works that remain
    separable from, or merely link (or bind by name) to the interfaces of,
    the Work and Derivative Works thereof.

    "Contribution" shall mean any work of authorship, including
    the original version of the Work and any modifications or additions
    to that Work or Derivative Works thereof, that is intentionally
    submitted to Licensor for inclusion in the Work by the copyright owner
    or by an individual or Legal Entity authorized to submit on behalf of
    the copyright owner. For the purposes of this definition, "submitted"
    means any form of electronic, verbal, or written communication sent
    to the Licensor or its representatives, including but not limited to
    communication on electronic mailing lists, source code control systems,
    and issue tracking systems that are managed by, or on behalf of, the
    Licensor for the purpose of discussing and improving the Work, but
    excluding communication that is conspicuously marked or otherwise
    designated in writing by the copyright owner as "Not a Contribution."

    "Contributor" shall mean Licensor and any individual or Legal Entity
    on behalf of whom a Contribution has been received by Licensor and
    subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
    this License, each Contributor hereby grants to You a perpetual,
    worldwide, non-exclusive, no-charge, royalty-free, irrevocable
    copyright license to reproduce, prepare Derivative Works of,
    publicly display, publicly perform, sublicense, and distribute the
    Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
    this License, each Contributor hereby grants to You a perpetual,
    worldwide, non-exclusive, no-charge, royalty-free, irrevocable
    (except as stated in this section) patent license to make, have made,
    use, offer to sell, sell, import, and otherwise transfer the Work,
    where such license applies only to those patent claims licensable
    by such Contributor that are necessarily infringed by their
    Contribution(s) alone or by combination of their Contribution(s)
    with the Work to which such Contribution(s) was submitted. If You
    institute patent litigation against any entity (including a
    cross-claim or counterclaim in a lawsuit) alleging that the Work
    or a Contribution incorporated within the Work constitutes direct
    or contributory patent infringement, then any patent licenses
    granted to You under this License for that Work shall terminate
    as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
    Work or Derivative Works thereof in any medium, with or without
    modifications, and in Source or Object form, provided that You
    meet the following conditions:
```

(continues on next page)

(continued from previous page)

```
    (a) You must give any other recipients of the Work or
        Derivative Works a copy of this License; and

    (b) You must cause any modified files to carry prominent notices
        stating that You changed the files; and

    (c) You must retain, in the Source form of any Derivative Works
        that You distribute, all copyright, patent, trademark, and
        attribution notices from the Source form of the Work,
        excluding those notices that do not pertain to any part of
        the Derivative Works; and

    (d) If the Work includes a "NOTICE" text file as part of its
        distribution, then any Derivative Works that You distribute must
        include a readable copy of the attribution notices contained
        within such NOTICE file, excluding those notices that do not
        pertain to any part of the Derivative Works, in at least one
        of the following places: within a NOTICE text file distributed
        as part of the Derivative Works; within the Source form or
        documentation, if provided along with the Derivative Works; or,
        within a display generated by the Derivative Works, if and
        wherever such third-party notices normally appear. The contents
        of the NOTICE file are for informational purposes only and
        do not modify the License. You may add Your own attribution
        notices within Derivative Works that You distribute, alongside
        or as an addendum to the NOTICE text from the Work, provided
        that such additional attribution notices cannot be construed
        as modifying the License.

    You may add Your own copyright statement to Your modifications and
    may provide additional or different license terms and conditions
    for use, reproduction, or distribution of Your modifications, or
    for any such Derivative Works as a whole, provided Your use,
    reproduction, and distribution of the Work otherwise complies with
    the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
   Notwithstanding the above, nothing herein shall supersede or modify
   the terms of any separate license agreement you may have executed
   with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
   names, trademarks, service marks, or product names of the Licensor,
   except as required for reasonable and customary use in describing the
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
```

(continues on next page)

(continued from previous page)

```
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
    implied, including, without limitation, any warranties or conditions
    of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
    PARTICULAR PURPOSE. You are solely responsible for determining the
    appropriateness of using or redistributing the Work and assume any
    risks associated with Your exercise of permissions under this License.

 8. Limitation of Liability. In no event and under no legal theory,
    whether in tort (including negligence), contract, or otherwise,
    unless required by applicable law (such as deliberate and grossly
    negligent acts) or agreed to in writing, shall any Contributor be
    liable to You for damages, including any direct, indirect, special,
    incidental, or consequential damages of any character arising as a
    result of this License or out of the use or inability to use the
    Work (including but not limited to damages for loss of goodwill,
    work stoppage, computer failure or malfunction, or any and all
    other commercial damages or losses), even if such Contributor
    has been advised of the possibility of such damages.

 9. Accepting Warranty or Additional Liability. While redistributing
    the Work or Derivative Works thereof, You may choose to offer,
    and charge a fee for, acceptance of support, warranty, indemnity,
    or other liability obligations and/or rights consistent with this
    License. However, in accepting such obligations, You may act only
    on Your own behalf and on Your sole responsibility, not on behalf
    of any other Contributor, and only if You agree to indemnify,
    defend, and hold each Contributor harmless for any liability
    incurred by, or claims asserted against, such Contributor by reason
    of your accepting any such warranty or additional liability.
```

14.19.2 Other Licenses

14.20 sniffio (1.3.0)

14.20.1 Declared Licenses

BSD-3-Clause, MIT, Apache-2.0

```
Copyright (c) <year> <owner> . All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

 1. Redistributions of source code must retain the above copyright notice,
    this list of conditions and the following disclaimer.

 2. Redistributions in binary form must reproduce the above copyright notice,
```

(continues on next page)

(continued from previous page)

```
      this list of conditions and the following disclaimer in the documentation
      and/or other materials provided with the distribution.

   3. Neither the name of the copyright holder nor the names of its
      contributors may be used to endorse or promote products derived from
      this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

```
The MIT License (MIT)

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

```
This software is made available under the terms of *either* of the
licenses found in LICENSE.APACHE2 or LICENSE.MIT. Contributions to are
made under the terms of *both* these licenses.
```

14.20.2 Other Licenses

14.21 tomli (2.0.1)

14.21.1 Declared Licenses

MIT

```
MIT License

Copyright (c) 2021 Taneli Hukkinen

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

14.21.2 Other Licenses

14.22 typing-extensions (4.8.0)

14.22.1 Declared Licenses

Python-2.0, 0BSD

```
1\. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the␣
↪Individual or Organization ("Licensee") accessing and otherwise using this software (
↪"Python") in source or binary form and its associated documentation.
   2. Subject to the terms and conditions of this License Agreement, PSF hereby grants␣
↪Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test,␣
↪perform and/or display publicly, prepare derivative works, distribute, and otherwise␣
↪use Python alone or in any derivative version, provided, however, that PSF's License␣
```

(continues on next page)

(continued from previous page)

```
↪Agreement and PSF's notice of copyright, i.e., "Copyright (c) 2001, 2002, 2003, 2004,␣
↪2005, 2006 Python Software Foundation; All Rights Reserved" are retained in Python␣
↪alone or in any derivative version prepared by Licensee.
   3. In the event Licensee prepares a derivative work that is based on or incorporates␣
↪Python or any part thereof, and wants to make the derivative work available to others␣
↪as provided herein, then Licensee hereby agrees to include in any such work a brief␣
↪summary of the changes made to Python.
   4. PSF is making Python available to Licensee on an "AS IS" basis. PSF MAKES NO␣
↪REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT␣
↪LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF␣
↪MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON WILL␣
↪NOT INFRINGE ANY THIRD PARTY RIGHTS.
   5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON FOR ANY␣
↪INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING,␣
↪DISTRIBUTING, OR OTHERWISE USING PYTHON, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF␣
↪THE POSSIBILITY THEREOF.
   6. This License Agreement will automatically terminate upon a material breach of its␣
↪terms and conditions.
   7. Nothing in this License Agreement shall be deemed to create any relationship of␣
↪agency, partnership, or joint venture between PSF and Licensee. This License Agreement␣
↪does not grant permission to use PSF trademarks or trade name in a trademark sense to␣
↪endorse or promote products or services of Licensee, or any third party.
   8. By copying, installing or otherwise using Python, Licensee agrees to be bound by␣
↪the terms and conditions of this License Agreement.
   1. This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an office at 160␣
↪Saratoga Avenue, Santa Clara, CA 95051, and the Individual or Organization ("Licensee
↪") accessing and otherwise using this software in source or binary form and its␣
↪associated documentation ("the Software").
   2. Subject to the terms and conditions of this BeOpen Python License Agreement,␣
↪BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license to␣
↪reproduce, analyze, test, perform and/or display publicly, prepare derivative works,␣
↪distribute, and otherwise use the Software alone or in any derivative version,␣
↪provided, however, that the BeOpen Python License is retained in the Software, alone␣
↪or in any derivative version prepared by Licensee.
   3. BeOpen is making the Software available to Licensee on an "AS IS" basis. BEOPEN␣
↪MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT␣
↪LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF␣
↪MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE␣
↪WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
   4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY␣
↪INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING, MODIFYING␣
↪OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE␣
↪POSSIBILITY THEREOF.
   5. This License Agreement will automatically terminate upon a material breach of its␣
↪terms and conditions.
   6. This License Agreement shall be governed by and interpreted in all respects by the␣
↪law of the State of California, excluding conflict of law provisions. Nothing in this␣
↪License Agreement shall be deemed to create any relationship of agency, partnership,␣
↪or joint venture between BeOpen and Licensee. This License Agreement does not grant␣
↪permission to use BeOpen trademarks or trade names in a trademark sense to endorse or␣
↪promote products or services of Licensee, or any third party. As an exception, the
↪"BeOpen Python" logos available at http://www.pythonlabs.com/logos.html may be used␣
```

(continues on next page)

(continued from previous page)

```
↪according to the permissions granted on that web page.
   7. By copying, installing or otherwise using the software, Licensee agrees to be␣
↪bound by the terms and conditions of this License Agreement.
   1. This LICENSE AGREEMENT is between the Corporation for National Research␣
↪Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191 ("CNRI"),␣
↪and the Individual or Organization ("Licensee") accessing and otherwise using Python 1.
↪6, beta 1 software in source or binary form and its associated documentation, as␣
↪released at the www.python.org Internet site on August 4, 2000 ("Python 1.6b1").
   2. Subject to the terms and conditions of this License Agreement, CNRI hereby grants␣
↪Licensee a non-exclusive, royalty-free, world-wide license to reproduce, analyze, test,
↪ perform and/or display publicly, prepare derivative works, distribute, and otherwise␣
↪use Python 1.6b1 alone or in any derivative version, provided, however, that CNRIs␣
↪License Agreement is retained in Python 1.6b1, alone or in any derivative version␣
↪prepared by Licensee.
   Alternately, in lieu of CNRIs License Agreement, Licensee may substitute the␣
↪following text (omitting the quotes): "Python 1.6, beta 1, is made available subject␣
↪to the terms and conditions in CNRIs License Agreement. This Agreement may be located␣
↪on the Internet using the following unique, persistent identifier (known as a handle):␣
↪1895.22/1011. This Agreement may also be obtained from a proxy server on the Internet␣
↪using the URL:http://hdl.handle.net/1895.22/1011".
   3. In the event Licensee prepares a derivative work that is based on or incorporates␣
↪Python 1.6b1 or any part thereof, and wants to make the derivative work available to␣
↪the public as provided herein, then Licensee hereby agrees to indicate in any such␣
↪work the nature of the modifications made to Python 1.6b1.
   4. CNRI is making Python 1.6b1 available to Licensee on an "AS IS" basis. CNRI MAKES␣
↪NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT␣
↪LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF␣
↪MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6b1␣
↪WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
   5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY␣
↪INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING, MODIFYING␣
↪OR DISTRIBUTING PYTHON 1.6b1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE␣
↪POSSIBILITY THEREOF.
   6. This License Agreement will automatically terminate upon a material breach of its␣
↪terms and conditions.
   7. This License Agreement shall be governed by and interpreted in all respects by the␣
↪law of the State of Virginia, excluding conflict of law provisions. Nothing in this␣
↪License Agreement shall be deemed to create any relationship of agency, partnership,␣
↪or joint venture between CNRI and Licensee. This License Agreement does not grant␣
↪permission to use CNRI trademarks or trade name in a trademark sense to endorse or␣
↪promote products or services of Licensee, or any third party.
   8. By clicking on the "ACCEPT" button where indicated, or by copying, installing or␣
↪otherwise using Python 1.6b1, Licensee agrees to be bound by the terms and conditions␣
↪of this License Agreement.
Copyright (c) 1991 - 1995, Stichting Mathematisch Centrum Amsterdam, The Netherlands.␣
↪All rights reserved.
Permission to use, copy, modify, and distribute this software and its documentation for␣
↪any purpose and without fee is hereby granted, provided that the above copyright␣
↪notice appear in all copies and that both that copyright notice and this permission␣
↪notice appear in supporting documentation, and that the name of Stichting Mathematisch␣
↪Centrum or CWI not be used in advertising or publicity pertaining to distribution of␣
↪the software without specific, written prior permission.
```

(continues on next page)

(continued from previous page)

```
STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,␣
↪INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL␣
↪STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL␣
↪DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER␣
↪IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN␣
↪CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
```

```
A. HISTORY OF THE SOFTWARE
==========================

Python was created in the early 1990s by Guido van Rossum at Stichting
Mathematisch Centrum (CWI, see https://www.cwi.nl) in the Netherlands
as a successor of a language called ABC.  Guido remains Python's
principal author, although it includes many contributions from others.

In 1995, Guido continued his work on Python at the Corporation for
National Research Initiatives (CNRI, see https://www.cnri.reston.va.us)
in Reston, Virginia where he released several versions of the
software.

In May 2000, Guido and the Python core development team moved to
BeOpen.com to form the BeOpen PythonLabs team.  In October of the same
year, the PythonLabs team moved to Digital Creations, which became
Zope Corporation.  In 2001, the Python Software Foundation (PSF, see
https://www.python.org/psf/) was formed, a non-profit organization
created specifically to own Python-related Intellectual Property.
Zope Corporation was a sponsoring member of the PSF.

All Python releases are Open Source (see https://opensource.org for
the Open Source Definition).  Historically, most, but not all, Python
releases have also been GPL-compatible; the table below summarizes
the various releases.

    Release         Derived     Year        Owner       GPL-
                    from                                compatible? (1)

    0.9.0 thru 1.2              1991-1995   CWI         yes
    1.3 thru 1.5.2  1.2         1995-1999   CNRI        yes
    1.6             1.5.2       2000        CNRI        no
    2.0             1.6         2000        BeOpen.com  no
    1.6.1           1.6         2001        CNRI        yes (2)
    2.1             2.0+1.6.1   2001        PSF         no
    2.0.1           2.0+1.6.1   2001        PSF         yes
    2.1.1           2.1+2.0.1   2001        PSF         yes
    2.1.2           2.1.1       2002        PSF         yes
    2.1.3           2.1.2       2002        PSF         yes
    2.2 and above   2.1.1       2001-now    PSF         yes

Footnotes:

(1) GPL-compatible doesn't mean that we're distributing Python under
    the GPL.  All Python licenses, unlike the GPL, let you distribute
```

(continues on next page)

(continued from previous page)

```
    a modified version without making your changes open source.  The
    GPL-compatible licenses make it possible to combine Python with
    other software that is released under the GPL; the others don't.

(2) According to Richard Stallman, 1.6.1 is not GPL-compatible,
    because its license has a choice of law clause.  According to
    CNRI, however, Stallman's lawyer has told CNRI's lawyer that 1.6.1
    is "not incompatible" with the GPL.

Thanks to the many outside volunteers who have worked under Guido's
direction to make these releases possible.


B. TERMS AND CONDITIONS FOR ACCESSING OR OTHERWISE USING PYTHON
===============================================================

Python software and documentation are licensed under the
Python Software Foundation License Version 2.

Starting with Python 3.8.6, examples, recipes, and other code in
the documentation are dual licensed under the PSF License Version 2
and the Zero-Clause BSD license.

Some software incorporated into Python is under different licenses.
The licenses are listed with code falling under that license.


PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2
--------------------------------------------

1\. This LICENSE AGREEMENT is between the Python Software Foundation
("PSF"), and the Individual or Organization ("Licensee") accessing and
otherwise using this software ("Python") in source or binary form and
its associated documentation.

2\. Subject to the terms and conditions of this License Agreement, PSF hereby
grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce,
analyze, test, perform and/or display publicly, prepare derivative works,
distribute, and otherwise use Python alone or in any derivative version,
provided, however, that PSF's License Agreement and PSF's notice of copyright,
i.e., "Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010,
2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023 Python␣
↪Software Foundation;
All Rights Reserved" are retained in Python alone or in any derivative version
prepared by Licensee.

3\. In the event Licensee prepares a derivative work that is based on
or incorporates Python or any part thereof, and wants to make
the derivative work available to others as provided herein, then
Licensee hereby agrees to include in any such work a brief summary of
the changes made to Python.
```

(continues on next page)

(continued from previous page)

```
4\. PSF is making Python available to Licensee on an "AS IS"
basis.  PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR
IMPLIED.  BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND
DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS
FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON WILL NOT
INFRINGE ANY THIRD PARTY RIGHTS.

5\. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON
FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS
A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON,
OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6\. This License Agreement will automatically terminate upon a material
breach of its terms and conditions.

7\. Nothing in this License Agreement shall be deemed to create any
relationship of agency, partnership, or joint venture between PSF and
Licensee.  This License Agreement does not grant permission to use PSF
trademarks or trade name in a trademark sense to endorse or promote
products or services of Licensee, or any third party.

8\. By copying, installing or otherwise using Python, Licensee
agrees to be bound by the terms and conditions of this License
Agreement.


BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0
------------------------------------------

BEOPEN PYTHON OPEN SOURCE LICENSE AGREEMENT VERSION 1

1\. This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an
office at 160 Saratoga Avenue, Santa Clara, CA 95051, and the
Individual or Organization ("Licensee") accessing and otherwise using
this software in source or binary form and its associated
documentation ("the Software").

2\. Subject to the terms and conditions of this BeOpen Python License
Agreement, BeOpen hereby grants Licensee a non-exclusive,
royalty-free, world-wide license to reproduce, analyze, test, perform
and/or display publicly, prepare derivative works, distribute, and
otherwise use the Software alone or in any derivative version,
provided, however, that the BeOpen Python License is retained in the
Software, alone or in any derivative version prepared by Licensee.

3\. BeOpen is making the Software available to Licensee on an "AS IS"
basis.  BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR
IMPLIED.  BY WAY OF EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND
DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS
FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE WILL NOT
INFRINGE ANY THIRD PARTY RIGHTS.
```

(continues on next page)

(continued from previous page)

```
4\. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE
SOFTWARE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS
AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY
DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

5\. This License Agreement will automatically terminate upon a material
breach of its terms and conditions.

6\. This License Agreement shall be governed by and interpreted in all
respects by the law of the State of California, excluding conflict of
law provisions.  Nothing in this License Agreement shall be deemed to
create any relationship of agency, partnership, or joint venture
between BeOpen and Licensee.  This License Agreement does not grant
permission to use BeOpen trademarks or trade names in a trademark
sense to endorse or promote products or services of Licensee, or any
third party.  As an exception, the "BeOpen Python" logos available at
http://www.pythonlabs.com/logos.html may be used according to the
permissions granted on that web page.

7\. By copying, installing or otherwise using the software, Licensee
agrees to be bound by the terms and conditions of this License
Agreement.


CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1
---------------------------------------

1\. This LICENSE AGREEMENT is between the Corporation for National
Research Initiatives, having an office at 1895 Preston White Drive,
Reston, VA 20191 ("CNRI"), and the Individual or Organization
("Licensee") accessing and otherwise using Python 1.6.1 software in
source or binary form and its associated documentation.

2\. Subject to the terms and conditions of this License Agreement, CNRI
hereby grants Licensee a nonexclusive, royalty-free, world-wide
license to reproduce, analyze, test, perform and/or display publicly,
prepare derivative works, distribute, and otherwise use Python 1.6.1
alone or in any derivative version, provided, however, that CNRI's
License Agreement and CNRI's notice of copyright, i.e., "Copyright (c)
1995-2001 Corporation for National Research Initiatives; All Rights
Reserved" are retained in Python 1.6.1 alone or in any derivative
version prepared by Licensee.  Alternately, in lieu of CNRI's License
Agreement, Licensee may substitute the following text (omitting the
quotes): "Python 1.6.1 is made available subject to the terms and
conditions in CNRI's License Agreement.  This Agreement together with
Python 1.6.1 may be located on the internet using the following
unique, persistent identifier (known as a handle): 1895.22/1013.  This
Agreement may also be obtained from a proxy server on the internet
using the following URL: http://hdl.handle.net/1895.22/1013".

3\. In the event Licensee prepares a derivative work that is based on
or incorporates Python 1.6.1 or any part thereof, and wants to make
```

(continues on next page)

(continued from previous page)

```
the derivative work available to others as provided herein, then
Licensee hereby agrees to include in any such work a brief summary of
the changes made to Python 1.6.1.

4\. CNRI is making Python 1.6.1 available to Licensee on an "AS IS"
basis.  CNRI MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR
IMPLIED.  BY WAY OF EXAMPLE, BUT NOT LIMITATION, CNRI MAKES NO AND
DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS
FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6.1 WILL NOT
INFRINGE ANY THIRD PARTY RIGHTS.

5\. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON
1.6.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS
A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1,
OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6\. This License Agreement will automatically terminate upon a material
breach of its terms and conditions.

7\. This License Agreement shall be governed by the federal
intellectual property law of the United States, including without
limitation the federal copyright law, and, to the extent such
U.S. federal law does not apply, by the law of the Commonwealth of
Virginia, excluding Virginia's conflict of law provisions.
Notwithstanding the foregoing, with regard to derivative works based
on Python 1.6.1 that incorporate non-separable material that was
previously distributed under the GNU General Public License (GPL), the
law of the Commonwealth of Virginia shall govern this License
Agreement only as to issues arising under or with respect to
Paragraphs 4, 5, and 7 of this License Agreement.  Nothing in this
License Agreement shall be deemed to create any relationship of
agency, partnership, or joint venture between CNRI and Licensee.  This
License Agreement does not grant permission to use CNRI trademarks or
trade name in a trademark sense to endorse or promote products or
services of Licensee, or any third party.

8\. By clicking on the "ACCEPT" button where indicated, or by copying,
installing or otherwise using Python 1.6.1, Licensee agrees to be
bound by the terms and conditions of this License Agreement.

        ACCEPT


CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2
--------------------------------------------------

Copyright (c) 1991 - 1995, Stichting Mathematisch Centrum Amsterdam,
The Netherlands.  All rights reserved.

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appear in all copies and that
```

(continues on next page)

(continued from previous page)

```
both that copyright notice and this permission notice appear in
supporting documentation, and that the name of Stichting Mathematisch
Centrum or CWI not be used in advertising or publicity pertaining to
distribution of the software without specific, written prior
permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO
THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS, IN NO EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE
FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT
OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

ZERO-CLAUSE BSD LICENSE FOR CODE IN THE PYTHON DOCUMENTATION
----------------------------------------------------------------------

Permission to use, copy, modify, and/or distribute this software for any
purpose with or without fee is hereby granted.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH
REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT,
INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM
LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
```

14.22.2 Other Licenses

14.23 urllib3 (2.1.0)

14.23.1 Declared Licenses

MIT

```
MIT License

Copyright (c) 2008-2020 Andrey Petrov and contributors.

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:
```

(continues on next page)

(continued from previous page)

```
The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

14.23.2 Other Licenses

14.24 watchfiles (0.21.0)

14.24.1 Declared Licenses

MIT

```
The MIT License (MIT)

Copyright (c) 2017, 2018, 2019, 2020, 2021, 2022 Samuel Colvin

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

14.24.2 Other Licenses

14.25 win32-setctime (1.1.0)

14.25.1 Declared Licenses

MIT

```
MIT License

Copyright (c) 2019 Delgan

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

14.25.2 Other Licenses

14.26 yarl (1.9.2)

14.26.1 Declared Licenses

Apache-2.0

```
                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

   TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION
```

(continues on next page)

(continued from previous page)

```
   1. Definitions.

      "License" shall mean the terms and conditions for use, reproduction,
      and distribution as defined by Sections 1 through 9 of this document.

      "Licensor" shall mean the copyright owner or entity authorized by
      the copyright owner that is granting the License.

      "Legal Entity" shall mean the union of the acting entity and all
      other entities that control, are controlled by, or are under common
      control with that entity. For the purposes of this definition,
      "control" means (i) the power, direct or indirect, to cause the
      direction or management of such entity, whether by contract or
      otherwise, or (ii) ownership of fifty percent (50%) or more of the
      outstanding shares, or (iii) beneficial ownership of such entity.

      "You" (or "Your") shall mean an individual or Legal Entity
      exercising permissions granted by this License.

      "Source" form shall mean the preferred form for making modifications,
      including but not limited to software source code, documentation
      source, and configuration files.

      "Object" form shall mean any form resulting from mechanical
      transformation or translation of a Source form, including but
      not limited to compiled object code, generated documentation,
      and conversions to other media types.

      "Work" shall mean the work of authorship, whether in Source or
      Object form, made available under the License, as indicated by a
      copyright notice that is included in or attached to the work
      (an example is provided in the Appendix below).

      "Derivative Works" shall mean any work, whether in Source or Object
      form, that is based on (or derived from) the Work and for which the
      editorial revisions, annotations, elaborations, or other modifications
      represent, as a whole, an original work of authorship. For the purposes
      of this License, Derivative Works shall not include works that remain
      separable from, or merely link (or bind by name) to the interfaces of,
      the Work and Derivative Works thereof.

      "Contribution" shall mean any work of authorship, including
      the original version of the Work and any modifications or additions
      to that Work or Derivative Works thereof, that is intentionally
      submitted to Licensor for inclusion in the Work by the copyright owner
      or by an individual or Legal Entity authorized to submit on behalf of
      the copyright owner. For the purposes of this definition, "submitted"
      means any form of electronic, verbal, or written communication sent
      to the Licensor or its representatives, including but not limited to
      communication on electronic mailing lists, source code control systems,
      and issue tracking systems that are managed by, or on behalf of, the
      Licensor for the purpose of discussing and improving the Work, but
```

(continues on next page)

(continued from previous page)

```
    excluding communication that is conspicuously marked or otherwise
    designated in writing by the copyright owner as "Not a Contribution."

    "Contributor" shall mean Licensor and any individual or Legal Entity
    on behalf of whom a Contribution has been received by Licensor and
    subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
    this License, each Contributor hereby grants to You a perpetual,
    worldwide, non-exclusive, no-charge, royalty-free, irrevocable
    copyright license to reproduce, prepare Derivative Works of,
    publicly display, publicly perform, sublicense, and distribute the
    Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
    this License, each Contributor hereby grants to You a perpetual,
    worldwide, non-exclusive, no-charge, royalty-free, irrevocable
    (except as stated in this section) patent license to make, have made,
    use, offer to sell, sell, import, and otherwise transfer the Work,
    where such license applies only to those patent claims licensable
    by such Contributor that are necessarily infringed by their
    Contribution(s) alone or by combination of their Contribution(s)
    with the Work to which such Contribution(s) was submitted. If You
    institute patent litigation against any entity (including a
    cross-claim or counterclaim in a lawsuit) alleging that the Work
    or a Contribution incorporated within the Work constitutes direct
    or contributory patent infringement, then any patent licenses
    granted to You under this License for that Work shall terminate
    as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
    Work or Derivative Works thereof in any medium, with or without
    modifications, and in Source or Object form, provided that You
    meet the following conditions:

    (a) You must give any other recipients of the Work or
        Derivative Works a copy of this License; and

    (b) You must cause any modified files to carry prominent notices
        stating that You changed the files; and

    (c) You must retain, in the Source form of any Derivative Works
        that You distribute, all copyright, patent, trademark, and
        attribution notices from the Source form of the Work,
        excluding those notices that do not pertain to any part of
        the Derivative Works; and

    (d) If the Work includes a "NOTICE" text file as part of its
        distribution, then any Derivative Works that You distribute must
        include a readable copy of the attribution notices contained
        within such NOTICE file, excluding those notices that do not
        pertain to any part of the Derivative Works, in at least one
```

(continues on next page)

(continued from previous page)

```
        of the following places: within a NOTICE text file distributed
        as part of the Derivative Works; within the Source form or
        documentation, if provided along with the Derivative Works; or,
        within a display generated by the Derivative Works, if and
        wherever such third-party notices normally appear. The contents
        of the NOTICE file are for informational purposes only and
        do not modify the License. You may add Your own attribution
        notices within Derivative Works that You distribute, alongside
        or as an addendum to the NOTICE text from the Work, provided
        that such additional attribution notices cannot be construed
        as modifying the License.

    You may add Your own copyright statement to Your modifications and
    may provide additional or different license terms and conditions
    for use, reproduction, or distribution of Your modifications, or
    for any such Derivative Works as a whole, provided Your use,
    reproduction, and distribution of the Work otherwise complies with
    the conditions stated in this License.

  5. Submission of Contributions. Unless You explicitly state otherwise,
    any Contribution intentionally submitted for inclusion in the Work
    by You to the Licensor shall be under the terms and conditions of
    this License, without any additional terms or conditions.
    Notwithstanding the above, nothing herein shall supersede or modify
    the terms of any separate license agreement you may have executed
    with Licensor regarding such Contributions.

  6. Trademarks. This License does not grant permission to use the trade
    names, trademarks, service marks, or product names of the Licensor,
    except as required for reasonable and customary use in describing the
    origin of the Work and reproducing the content of the NOTICE file.

  7. Disclaimer of Warranty. Unless required by applicable law or
    agreed to in writing, Licensor provides the Work (and each
    Contributor provides its Contributions) on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
    implied, including, without limitation, any warranties or conditions
    of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
    PARTICULAR PURPOSE. You are solely responsible for determining the
    appropriateness of using or redistributing the Work and assume any
    risks associated with Your exercise of permissions under this License.

  8. Limitation of Liability. In no event and under no legal theory,
    whether in tort (including negligence), contract, or otherwise,
    unless required by applicable law (such as deliberate and grossly
    negligent acts) or agreed to in writing, shall any Contributor be
    liable to You for damages, including any direct, indirect, special,
    incidental, or consequential damages of any character arising as a
    result of this License or out of the use or inability to use the
    Work (including but not limited to damages for loss of goodwill,
    work stoppage, computer failure or malfunction, or any and all
    other commercial damages or losses), even if such Contributor
```

(continues on next page)

(continued from previous page)

```
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets "[]"
   replaced with your own identifying information. (Don't include
   the brackets!)  The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
   same "printed page" as the copyright notice for easier
   identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

14.26.2 Other Licenses

CHAPTER

FIFTEEN

LICENSES

15.1 Apache-2.0

```
                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

   TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

   1. Definitions.

      &quot;License&quot; shall mean the terms and conditions for use, reproduction,
      and distribution as defined by Sections 1 through 9 of this document.

      &quot;Licensor&quot; shall mean the copyright owner or entity authorized by
      the copyright owner that is granting the License.

      &quot;Legal Entity&quot; shall mean the union of the acting entity and all
      other entities that control, are controlled by, or are under common
      control with that entity. For the purposes of this definition,
      &quot;control&quot; means (i) the power, direct or indirect, to cause the
      direction or management of such entity, whether by contract or
      otherwise, or (ii) ownership of fifty percent (50%) or more of the
      outstanding shares, or (iii) beneficial ownership of such entity.

      &quot;You&quot; (or &quot;Your&quot;) shall mean an individual or Legal Entity
      exercising permissions granted by this License.

      &quot;Source&quot; form shall mean the preferred form for making modifications,
      including but not limited to software source code, documentation
      source, and configuration files.

      &quot;Object&quot; form shall mean any form resulting from mechanical
      transformation or translation of a Source form, including but
      not limited to compiled object code, generated documentation,
      and conversions to other media types.

      &quot;Work&quot; shall mean the work of authorship, whether in Source or
      Object form, made available under the License, as indicated by a
```

(continues on next page)

(continued from previous page)

```
    copyright notice that is included in or attached to the work
    (an example is provided in the Appendix below).

    &quot;Derivative Works&quot; shall mean any work, whether in Source or Object
    form, that is based on (or derived from) the Work and for which the
    editorial revisions, annotations, elaborations, or other modifications
    represent, as a whole, an original work of authorship. For the purposes
    of this License, Derivative Works shall not include works that remain
    separable from, or merely link (or bind by name) to the interfaces of,
    the Work and Derivative Works thereof.

    &quot;Contribution&quot; shall mean any work of authorship, including
    the original version of the Work and any modifications or additions
    to that Work or Derivative Works thereof, that is intentionally
    submitted to Licensor for inclusion in the Work by the copyright owner
    or by an individual or Legal Entity authorized to submit on behalf of
    the copyright owner. For the purposes of this definition, &quot;submitted&quot;
    means any form of electronic, verbal, or written communication sent
    to the Licensor or its representatives, including but not limited to
    communication on electronic mailing lists, source code control systems,
    and issue tracking systems that are managed by, or on behalf of, the
    Licensor for the purpose of discussing and improving the Work, but
    excluding communication that is conspicuously marked or otherwise
    designated in writing by the copyright owner as &quot;Not a Contribution.&quot;

    &quot;Contributor&quot; shall mean Licensor and any individual or Legal Entity
    on behalf of whom a Contribution has been received by Licensor and
    subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
    this License, each Contributor hereby grants to You a perpetual,
    worldwide, non-exclusive, no-charge, royalty-free, irrevocable
    copyright license to reproduce, prepare Derivative Works of,
    publicly display, publicly perform, sublicense, and distribute the
    Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
    this License, each Contributor hereby grants to You a perpetual,
    worldwide, non-exclusive, no-charge, royalty-free, irrevocable
    (except as stated in this section) patent license to make, have made,
    use, offer to sell, sell, import, and otherwise transfer the Work,
    where such license applies only to those patent claims licensable
    by such Contributor that are necessarily infringed by their
    Contribution(s) alone or by combination of their Contribution(s)
    with the Work to which such Contribution(s) was submitted. If You
    institute patent litigation against any entity (including a
    cross-claim or counterclaim in a lawsuit) alleging that the Work
    or a Contribution incorporated within the Work constitutes direct
    or contributory patent infringement, then any patent licenses
    granted to You under this License for that Work shall terminate
    as of the date such litigation is filed.
```

(continues on next page)

(continued from previous page)

```
   4. Redistribution. You may reproduce and distribute copies of the
      Work or Derivative Works thereof in any medium, with or without
      modifications, and in Source or Object form, provided that You
      meet the following conditions:

      (a) You must give any other recipients of the Work or
          Derivative Works a copy of this License; and

      (b) You must cause any modified files to carry prominent notices
          stating that You changed the files; and

      (c) You must retain, in the Source form of any Derivative Works
          that You distribute, all copyright, patent, trademark, and
          attribution notices from the Source form of the Work,
          excluding those notices that do not pertain to any part of
          the Derivative Works; and

      (d) If the Work includes a &quot;NOTICE&quot; text file as part of its
          distribution, then any Derivative Works that You distribute must
          include a readable copy of the attribution notices contained
          within such NOTICE file, excluding those notices that do not
          pertain to any part of the Derivative Works, in at least one
          of the following places: within a NOTICE text file distributed
          as part of the Derivative Works; within the Source form or
          documentation, if provided along with the Derivative Works; or,
          within a display generated by the Derivative Works, if and
          wherever such third-party notices normally appear. The contents
          of the NOTICE file are for informational purposes only and
          do not modify the License. You may add Your own attribution
          notices within Derivative Works that You distribute, alongside
          or as an addendum to the NOTICE text from the Work, provided
          that such additional attribution notices cannot be construed
          as modifying the License.

      You may add Your own copyright statement to Your modifications and
      may provide additional or different license terms and conditions
      for use, reproduction, or distribution of Your modifications, or
      for any such Derivative Works as a whole, provided Your use,
      reproduction, and distribution of the Work otherwise complies with
      the conditions stated in this License.

   5. Submission of Contributions. Unless You explicitly state otherwise,
      any Contribution intentionally submitted for inclusion in the Work
      by You to the Licensor shall be under the terms and conditions of
      this License, without any additional terms or conditions.
      Notwithstanding the above, nothing herein shall supersede or modify
      the terms of any separate license agreement you may have executed
      with Licensor regarding such Contributions.

   6. Trademarks. This License does not grant permission to use the trade
      names, trademarks, service marks, or product names of the Licensor,
      except as required for reasonable and customary use in describing the
```

(continues on next page)

(continued from previous page)

```
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an &quot;AS IS&quot; BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.
```

15.2 Apache-2.0

```
                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   &quot;License&quot; shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.
```

(continues on next page)

(continued from previous page)

```
      &quot;Licensor&quot; shall mean the copyright owner or entity authorized by
      the copyright owner that is granting the License.

      &quot;Legal Entity&quot; shall mean the union of the acting entity and all
      other entities that control, are controlled by, or are under common
      control with that entity. For the purposes of this definition,
      &quot;control&quot; means (i) the power, direct or indirect, to cause the
      direction or management of such entity, whether by contract or
      otherwise, or (ii) ownership of fifty percent (50%) or more of the
      outstanding shares, or (iii) beneficial ownership of such entity.

      &quot;You&quot; (or &quot;Your&quot;) shall mean an individual or Legal Entity
      exercising permissions granted by this License.

      &quot;Source&quot; form shall mean the preferred form for making modifications,
      including but not limited to software source code, documentation
      source, and configuration files.

      &quot;Object&quot; form shall mean any form resulting from mechanical
      transformation or translation of a Source form, including but
      not limited to compiled object code, generated documentation,
      and conversions to other media types.

      &quot;Work&quot; shall mean the work of authorship, whether in Source or
      Object form, made available under the License, as indicated by a
      copyright notice that is included in or attached to the work
      (an example is provided in the Appendix below).

      &quot;Derivative Works&quot; shall mean any work, whether in Source or Object
      form, that is based on (or derived from) the Work and for which the
      editorial revisions, annotations, elaborations, or other modifications
      represent, as a whole, an original work of authorship. For the purposes
      of this License, Derivative Works shall not include works that remain
      separable from, or merely link (or bind by name) to the interfaces of,
      the Work and Derivative Works thereof.

      &quot;Contribution&quot; shall mean any work of authorship, including
      the original version of the Work and any modifications or additions
      to that Work or Derivative Works thereof, that is intentionally
      submitted to Licensor for inclusion in the Work by the copyright owner
      or by an individual or Legal Entity authorized to submit on behalf of
      the copyright owner. For the purposes of this definition, &quot;submitted&quot;
      means any form of electronic, verbal, or written communication sent
      to the Licensor or its representatives, including but not limited to
      communication on electronic mailing lists, source code control systems,
      and issue tracking systems that are managed by, or on behalf of, the
      Licensor for the purpose of discussing and improving the Work, but
      excluding communication that is conspicuously marked or otherwise
      designated in writing by the copyright owner as &quot;Not a Contribution.&quot;

      &quot;Contributor&quot; shall mean Licensor and any individual or Legal Entity
      on behalf of whom a Contribution has been received by Licensor and
```

(continues on next page)

(continued from previous page)

```
    subsequently incorporated within the Work.

 2. Grant of Copyright License. Subject to the terms and conditions of
    this License, each Contributor hereby grants to You a perpetual,
    worldwide, non-exclusive, no-charge, royalty-free, irrevocable
    copyright license to reproduce, prepare Derivative Works of,
    publicly display, publicly perform, sublicense, and distribute the
    Work and such Derivative Works in Source or Object form.

 3. Grant of Patent License. Subject to the terms and conditions of
    this License, each Contributor hereby grants to You a perpetual,
    worldwide, non-exclusive, no-charge, royalty-free, irrevocable
    (except as stated in this section) patent license to make, have made,
    use, offer to sell, sell, import, and otherwise transfer the Work,
    where such license applies only to those patent claims licensable
    by such Contributor that are necessarily infringed by their
    Contribution(s) alone or by combination of their Contribution(s)
    with the Work to which such Contribution(s) was submitted. If You
    institute patent litigation against any entity (including a
    cross-claim or counterclaim in a lawsuit) alleging that the Work
    or a Contribution incorporated within the Work constitutes direct
    or contributory patent infringement, then any patent licenses
    granted to You under this License for that Work shall terminate
    as of the date such litigation is filed.

 4. Redistribution. You may reproduce and distribute copies of the
    Work or Derivative Works thereof in any medium, with or without
    modifications, and in Source or Object form, provided that You
    meet the following conditions:

    (a) You must give any other recipients of the Work or
        Derivative Works a copy of this License; and

    (b) You must cause any modified files to carry prominent notices
        stating that You changed the files; and

    (c) You must retain, in the Source form of any Derivative Works
        that You distribute, all copyright, patent, trademark, and
        attribution notices from the Source form of the Work,
        excluding those notices that do not pertain to any part of
        the Derivative Works; and

    (d) If the Work includes a &quot;NOTICE&quot; text file as part of its
        distribution, then any Derivative Works that You distribute must
        include a readable copy of the attribution notices contained
        within such NOTICE file, excluding those notices that do not
        pertain to any part of the Derivative Works, in at least one
        of the following places: within a NOTICE text file distributed
        as part of the Derivative Works; within the Source form or
        documentation, if provided along with the Derivative Works; or,
        within a display generated by the Derivative Works, if and
        wherever such third-party notices normally appear. The contents
```

(continues on next page)

(continued from previous page)

```
        of the NOTICE file are for informational purposes only and
        do not modify the License. You may add Your own attribution
        notices within Derivative Works that You distribute, alongside
        or as an addendum to the NOTICE text from the Work, provided
        that such additional attribution notices cannot be construed
        as modifying the License.

    You may add Your own copyright statement to Your modifications and
    may provide additional or different license terms and conditions
    for use, reproduction, or distribution of Your modifications, or
    for any such Derivative Works as a whole, provided Your use,
    reproduction, and distribution of the Work otherwise complies with
    the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
    any Contribution intentionally submitted for inclusion in the Work
    by You to the Licensor shall be under the terms and conditions of
    this License, without any additional terms or conditions.
    Notwithstanding the above, nothing herein shall supersede or modify
    the terms of any separate license agreement you may have executed
    with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
    names, trademarks, service marks, or product names of the Licensor,
    except as required for reasonable and customary use in describing the
    origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
    agreed to in writing, Licensor provides the Work (and each
    Contributor provides its Contributions) on an &quot;AS IS&quot; BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
    implied, including, without limitation, any warranties or conditions
    of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
    PARTICULAR PURPOSE. You are solely responsible for determining the
    appropriateness of using or redistributing the Work and assume any
    risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
    whether in tort (including negligence), contract, or otherwise,
    unless required by applicable law (such as deliberate and grossly
    negligent acts) or agreed to in writing, shall any Contributor be
    liable to You for damages, including any direct, indirect, special,
    incidental, or consequential damages of any character arising as a
    result of this License or out of the use or inability to use the
    Work (including but not limited to damages for loss of goodwill,
    work stoppage, computer failure or malfunction, or any and all
    other commercial damages or losses), even if such Contributor
    has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
    the Work or Derivative Works thereof, You may choose to offer,
    and charge a fee for, acceptance of support, warranty, indemnity,
```

(continues on next page)

(continued from previous page)

```
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets &quot;[]&quot;
   replaced with your own identifying information. (Don&#39;t include
   the brackets!)  The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
   same &quot;printed page&quot; as the copyright notice for easier
   identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the &quot;License&quot;);
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an &quot;AS IS&quot; BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

15.3 Apache-2.0

```
Copyright 2016-2021 Andrew Svetlov and aio-libs team

Licensed under the Apache License, Version 2.0 (the &quot;License&quot;);
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an &quot;AS IS&quot; BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
```

(continues on next page)

(continued from previous page)

```
   limitations under the License.
```

15.4 Apache-2.0

```
   Copyright aio-libs contributors.

   Licensed under the Apache License, Version 2.0 (the &quot;License&quot;);
   you may not use this file except in compliance with the License.
   You may obtain a copy of the License at

       http://www.apache.org/licenses/LICENSE-2.0

   Unless required by applicable law or agreed to in writing, software
   distributed under the License is distributed on an &quot;AS IS&quot; BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   See the License for the specific language governing permissions and
   limitations under the License.
```

15.5 0BSD

```
A. HISTORY OF THE SOFTWARE
==========================

Python was created in the early 1990s by Guido van Rossum at Stichting
Mathematisch Centrum (CWI, see https://www.cwi.nl) in the Netherlands
as a successor of a language called ABC.  Guido remains Python&#39;s
principal author, although it includes many contributions from others.

In 1995, Guido continued his work on Python at the Corporation for
National Research Initiatives (CNRI, see https://www.cnri.reston.va.us)
in Reston, Virginia where he released several versions of the
software.

In May 2000, Guido and the Python core development team moved to
BeOpen.com to form the BeOpen PythonLabs team.  In October of the same
year, the PythonLabs team moved to Digital Creations, which became
Zope Corporation.  In 2001, the Python Software Foundation (PSF, see
https://www.python.org/psf/) was formed, a non-profit organization
created specifically to own Python-related Intellectual Property.
Zope Corporation was a sponsoring member of the PSF.

All Python releases are Open Source (see https://opensource.org for
the Open Source Definition).  Historically, most, but not all, Python
releases have also been GPL-compatible; the table below summarizes
the various releases.
```

(continues on next page)

(continued from previous page)

```
    Release         Derived     Year        Owner       GPL-
                    from                                compatible? (1)

    0.9.0 thru 1.2              1991-1995   CWI         yes
    1.3 thru 1.5.2  1.2         1995-1999   CNRI        yes
    1.6             1.5.2       2000        CNRI        no
    2.0             1.6         2000        BeOpen.com  no
    1.6.1           1.6         2001        CNRI        yes (2)
    2.1             2.0+1.6.1   2001        PSF         no
    2.0.1           2.0+1.6.1   2001        PSF         yes
    2.1.1           2.1+2.0.1   2001        PSF         yes
    2.1.2           2.1.1       2002        PSF         yes
    2.1.3           2.1.2       2002        PSF         yes
    2.2 and above   2.1.1       2001-now    PSF         yes

Footnotes:

(1) GPL-compatible doesn&#39;t mean that we&#39;re distributing Python under
    the GPL.  All Python licenses, unlike the GPL, let you distribute
    a modified version without making your changes open source.  The
    GPL-compatible licenses make it possible to combine Python with
    other software that is released under the GPL; the others don&#39;t.

(2) According to Richard Stallman, 1.6.1 is not GPL-compatible,
    because its license has a choice of law clause.  According to
    CNRI, however, Stallman&#39;s lawyer has told CNRI&#39;s lawyer that 1.6.1
    is &quot;not incompatible&quot; with the GPL.

Thanks to the many outside volunteers who have worked under Guido&#39;s
direction to make these releases possible.


B. TERMS AND CONDITIONS FOR ACCESSING OR OTHERWISE USING PYTHON
===============================================================

Python software and documentation are licensed under the
Python Software Foundation License Version 2.

Starting with Python 3.8.6, examples, recipes, and other code in
the documentation are dual licensed under the PSF License Version 2
and the Zero-Clause BSD license.

Some software incorporated into Python is under different licenses.
The licenses are listed with code falling under that license.


PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2
--------------------------------------------

1. This LICENSE AGREEMENT is between the Python Software Foundation
(&quot;PSF&quot;), and the Individual or Organization (&quot;Licensee&quot;) accessing␣
```

(continues on next page)

(continued from previous page)

```
↪and
otherwise using this software (&quot;Python&quot;) in source or binary form and
its associated documentation.

2. Subject to the terms and conditions of this License Agreement, PSF hereby
grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce,
analyze, test, perform and/or display publicly, prepare derivative works,
distribute, and otherwise use Python alone or in any derivative version,
provided, however, that PSF&#39;s License Agreement and PSF&#39;s notice of copyright,
i.e., &quot;Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010,
2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023 Python␣
↪Software Foundation;
All Rights Reserved&quot; are retained in Python alone or in any derivative version
prepared by Licensee.

3. In the event Licensee prepares a derivative work that is based on
or incorporates Python or any part thereof, and wants to make
the derivative work available to others as provided herein, then
Licensee hereby agrees to include in any such work a brief summary of
the changes made to Python.

4. PSF is making Python available to Licensee on an &quot;AS IS&quot;
basis.  PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR
IMPLIED.  BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND
DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS
FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON WILL NOT
INFRINGE ANY THIRD PARTY RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON
FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS
A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON,
OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material
breach of its terms and conditions.

7. Nothing in this License Agreement shall be deemed to create any
relationship of agency, partnership, or joint venture between PSF and
Licensee.  This License Agreement does not grant permission to use PSF
trademarks or trade name in a trademark sense to endorse or promote
products or services of Licensee, or any third party.

8. By copying, installing or otherwise using Python, Licensee
agrees to be bound by the terms and conditions of this License
Agreement.


BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0
------------------------------------------

BEOPEN PYTHON OPEN SOURCE LICENSE AGREEMENT VERSION 1
```

(continues on next page)

(continued from previous page)

```
1. This LICENSE AGREEMENT is between BeOpen.com (&quot;BeOpen&quot;), having an
office at 160 Saratoga Avenue, Santa Clara, CA 95051, and the
Individual or Organization (&quot;Licensee&quot;) accessing and otherwise using
this software in source or binary form and its associated
documentation (&quot;the Software&quot;).

2. Subject to the terms and conditions of this BeOpen Python License
Agreement, BeOpen hereby grants Licensee a non-exclusive,
royalty-free, world-wide license to reproduce, analyze, test, perform
and/or display publicly, prepare derivative works, distribute, and
otherwise use the Software alone or in any derivative version,
provided, however, that the BeOpen Python License is retained in the
Software, alone or in any derivative version prepared by Licensee.

3. BeOpen is making the Software available to Licensee on an &quot;AS IS&quot;
basis.  BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR
IMPLIED.  BY WAY OF EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND
DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS
FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE WILL NOT
INFRINGE ANY THIRD PARTY RIGHTS.

4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE
SOFTWARE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS
AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY
DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

5. This License Agreement will automatically terminate upon a material
breach of its terms and conditions.

6. This License Agreement shall be governed by and interpreted in all
respects by the law of the State of California, excluding conflict of
law provisions.  Nothing in this License Agreement shall be deemed to
create any relationship of agency, partnership, or joint venture
between BeOpen and Licensee.  This License Agreement does not grant
permission to use BeOpen trademarks or trade names in a trademark
sense to endorse or promote products or services of Licensee, or any
third party.  As an exception, the &quot;BeOpen Python&quot; logos available at
http://www.pythonlabs.com/logos.html may be used according to the
permissions granted on that web page.

7. By copying, installing or otherwise using the software, Licensee
agrees to be bound by the terms and conditions of this License
Agreement.


CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1
---------------------------------------

1. This LICENSE AGREEMENT is between the Corporation for National
Research Initiatives, having an office at 1895 Preston White Drive,
Reston, VA 20191 (&quot;CNRI&quot;), and the Individual or Organization
(&quot;Licensee&quot;) accessing and otherwise using Python 1.6.1 software in
```

(continues on next page)

(continued from previous page)

```
source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, CNRI
hereby grants Licensee a nonexclusive, royalty-free, world-wide
license to reproduce, analyze, test, perform and/or display publicly,
prepare derivative works, distribute, and otherwise use Python 1.6.1
alone or in any derivative version, provided, however, that CNRI&#39;s
License Agreement and CNRI&#39;s notice of copyright, i.e., &quot;Copyright (c)
1995-2001 Corporation for National Research Initiatives; All Rights
Reserved&quot; are retained in Python 1.6.1 alone or in any derivative
version prepared by Licensee.  Alternately, in lieu of CNRI&#39;s License
Agreement, Licensee may substitute the following text (omitting the
quotes): &quot;Python 1.6.1 is made available subject to the terms and
conditions in CNRI&#39;s License Agreement.  This Agreement together with
Python 1.6.1 may be located on the internet using the following
unique, persistent identifier (known as a handle): 1895.22/1013.  This
Agreement may also be obtained from a proxy server on the internet
using the following URL: http://hdl.handle.net/1895.22/1013&quot;.

3. In the event Licensee prepares a derivative work that is based on
or incorporates Python 1.6.1 or any part thereof, and wants to make
the derivative work available to others as provided herein, then
Licensee hereby agrees to include in any such work a brief summary of
the changes made to Python 1.6.1.

4. CNRI is making Python 1.6.1 available to Licensee on an &quot;AS IS&quot;
basis.  CNRI MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR
IMPLIED.  BY WAY OF EXAMPLE, BUT NOT LIMITATION, CNRI MAKES NO AND
DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS
FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6.1 WILL NOT
INFRINGE ANY THIRD PARTY RIGHTS.

5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON
1.6.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS
A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1,
OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material
breach of its terms and conditions.

7. This License Agreement shall be governed by the federal
intellectual property law of the United States, including without
limitation the federal copyright law, and, to the extent such
U.S. federal law does not apply, by the law of the Commonwealth of
Virginia, excluding Virginia&#39;s conflict of law provisions.
Notwithstanding the foregoing, with regard to derivative works based
on Python 1.6.1 that incorporate non-separable material that was
previously distributed under the GNU General Public License (GPL), the
law of the Commonwealth of Virginia shall govern this License
Agreement only as to issues arising under or with respect to
Paragraphs 4, 5, and 7 of this License Agreement.  Nothing in this
License Agreement shall be deemed to create any relationship of
```

(continues on next page)

(continued from previous page)

```
agency, partnership, or joint venture between CNRI and Licensee.  This
License Agreement does not grant permission to use CNRI trademarks or
trade name in a trademark sense to endorse or promote products or
services of Licensee, or any third party.

8. By clicking on the &quot;ACCEPT&quot; button where indicated, or by copying,
installing or otherwise using Python 1.6.1, Licensee agrees to be
bound by the terms and conditions of this License Agreement.

        ACCEPT


CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2
--------------------------------------------------

Copyright (c) 1991 - 1995, Stichting Mathematisch Centrum Amsterdam,
The Netherlands.  All rights reserved.

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in
supporting documentation, and that the name of Stichting Mathematisch
Centrum or CWI not be used in advertising or publicity pertaining to
distribution of the software without specific, written prior
permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO
THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS, IN NO EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE
FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT
OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

ZERO-CLAUSE BSD LICENSE FOR CODE IN THE PYTHON DOCUMENTATION
----------------------------------------------------------------------

Permission to use, copy, modify, and/or distribute this software for any
purpose with or without fee is hereby granted.

THE SOFTWARE IS PROVIDED &quot;AS IS&quot; AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH
REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT,
INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM
LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
```

15.6 AGPL-3.0-only

```
GNU AFFERO GENERAL PUBLIC LICENSE
Version 3, 19 November 2007
Copyright (C) 2007 Free Software Foundation, Inc. &lt;http s ://fsf.org/&gt;
Everyone is permitted to copy and distribute verbatim copies of this license document,␣
↪but changing it is not allowed.
Preamble
The GNU Affero General Public License is a free, copyleft license for software and other␣
↪kinds of works, specifically designed to ensure cooperation with the community in the␣
↪case of network server software.
The licenses for most software and other practical works are designed to take away your␣
↪freedom to share and change the works. By contrast, our General Public Licenses are␣
↪intended to guarantee your freedom to share and change all versions of a program--to␣
↪make sure it remains free software for all its users.
When we speak of free software, we are referring to freedom, not price. Our General␣
↪Public Licenses are designed to make sure that you have the freedom to distribute␣
↪copies of free software (and charge for them if you wish), that you receive source␣
↪code or can get it if you want it, that you can change the software or use pieces of␣
↪it in new free programs, and that you know you can do these things.
Developers that use our General Public Licenses protect your rights with two steps: (1)␣
↪assert copyright on the software, and (2) offer you this License which gives you legal␣
↪permission to copy, distribute and/or modify the software.
A secondary benefit of defending all users&#39; freedom is that improvements made in␣
↪alternate versions of the program, if they receive widespread use, become available␣
↪for other developers to incorporate. Many developers of free software are heartened␣
↪and encouraged by the resulting cooperation. However, in the case of software used on␣
↪network servers, this result may fail to come about. The GNU General Public License␣
↪permits making a modified version and letting the public access it on a server without␣
↪ever releasing its source code to the public.
The GNU Affero General Public License is designed specifically to ensure that, in such␣
↪cases, the modified source code becomes available to the community. It requires the␣
↪operator of a network server to provide the source code of the modified version␣
↪running there to the users of that server. Therefore, public use of a modified version,
↪ on a publicly accessible server, gives the public access to the source code of the␣
↪modified version.
An older license, called the Affero General Public License and published by Affero, was␣
↪designed to accomplish similar goals. This is a different license, not a version of␣
↪the Affero GPL, but Affero has released a new version of the Affero GPL which permits␣
↪relicensing under this license.
The precise terms and conditions for copying, distribution and modification follow.
TERMS AND CONDITIONS
   0. Definitions.
   &quot;This License&quot; refers to version 3 of the GNU Affero General Public License.
   &quot;Copyright&quot; also means copyright-like laws that apply to other kinds of␣
↪works, such as semiconductor masks.
   &quot;The Program&quot; refers to any copyrightable work licensed under this License.␣
↪Each licensee is addressed as &quot;you&quot;. &quot;Licensees&quot; and &quot;
↪recipients&quot; may be individuals or organizations.
   To &quot;modify&quot; a work means to copy from or adapt all or part of the work in a␣
↪fashion requiring copyright permission, other than the making of an exact copy. The␣
↪resulting work is called a &quot;modified version&quot; of the earlier work or a work &
```

(continues on next page)

(continued from previous page)

```
↪quot;based on&quot; the earlier work.
  A &quot;covered work&quot; means either the unmodified Program or a work based on the␣
↪Program.
  To &quot;propagate&quot; a work means to do anything with it that, without permission,
↪ would make you directly or secondarily liable for infringement under applicable␣
↪copyright law, except executing it on a computer or modifying a private copy.␣
↪Propagation includes copying, distribution (with or without modification), making␣
↪available to the public, and in some countries other activities as well.
  To &quot;convey&quot; a work means any kind of propagation that enables other parties␣
↪to make or receive copies. Mere interaction with a user through a computer network,␣
↪with no transfer of a copy, is not conveying.
  An interactive user interface displays &quot;Appropriate Legal Notices&quot; to the␣
↪extent that it includes a convenient and prominently visible feature that (1) displays␣
↪an appropriate copyright notice, and (2) tells the user that there is no warranty for␣
↪the work (except to the extent that warranties are provided), that licensees may␣
↪convey the work under this License, and how to view a copy of this License. If the␣
↪interface presents a list of user commands or options, such as a menu, a prominent␣
↪item in the list meets this criterion.
  1. Source Code.
  The &quot;source code&quot; for a work means the preferred form of the work for␣
↪making modifications to it. &quot;Object code&quot; means any non-source form of a␣
↪work.
  A &quot;Standard Interface&quot; means an interface that either is an official␣
↪standard defined by a recognized standards body, or, in the case of interfaces␣
↪specified for a particular programming language, one that is widely used among␣
↪developers working in that language.
  The &quot;System Libraries&quot; of an executable work include anything, other than␣
↪the work as a whole, that (a) is included in the normal form of packaging a Major␣
↪Component, but which is not part of that Major Component, and (b) serves only to␣
↪enable use of the work with that Major Component, or to implement a Standard Interface␣
↪for which an implementation is available to the public in source code form. A &quot;
↪Major Component&quot;, in this context, means a major essential component (kernel,␣
↪window system, and so on) of the specific operating system (if any) on which the␣
↪executable work runs, or a compiler used to produce the work, or an object code␣
↪interpreter used to run it.
  The &quot;Corresponding Source&quot; for a work in object code form means all the␣
↪source code needed to generate, install, and (for an executable work) run the object␣
↪code and to modify the work, including scripts to control those activities. However,␣
↪it does not include the work&#39;s System Libraries, or general-purpose tools or␣
↪generally available free programs which are used unmodified in performing those␣
↪activities but which are not part of the work. For example, Corresponding Source␣
↪includes interface definition files associated with source files for the work, and the␣
↪source code for shared libraries and dynamically linked subprograms that the work is␣
↪specifically designed to require, such as by intimate data communication or control␣
↪flow between those
  subprograms and other parts of the work.
  The Corresponding Source need not include anything that users can regenerate␣
↪automatically from other parts of the Corresponding Source.
  The Corresponding Source for a work in source code form is that same work.
  2. Basic Permissions.
  All rights granted under this License are granted for the term of copyright on the␣
↪Program, and are irrevocable provided the stated conditions are met. This License␣
```

(continues on next page)

(continued from previous page)

```
↪explicitly affirms your unlimited permission to run the unmodified Program. The output␣
↪from running a covered work is covered by this License only if the output, given its␣
↪content, constitutes a covered work. This License acknowledges your rights of fair use␣
↪or other equivalent, as provided by copyright law.
  You may make, run and propagate covered works that you do not convey, without␣
↪conditions so long as your license otherwise remains in force. You may convey covered␣
↪works to others for the sole purpose of having them make modifications exclusively for␣
↪you, or provide you with facilities for running those works, provided that you comply␣
↪with the terms of this License in conveying all material for which you do not control␣
↪copyright. Those thus making or running the covered works for you must do so␣
↪exclusively on your behalf, under your direction and control, on terms that prohibit␣
↪them from making any copies of your copyrighted material outside their relationship␣
↪with you.
  Conveying under any other circumstances is permitted solely under the conditions␣
↪stated below. Sublicensing is not allowed; section 10 makes it unnecessary.
  3. Protecting Users&amp;#39; Legal Rights From Anti-Circumvention Law.
  No covered work shall be deemed part of an effective technological measure under any␣
↪applicable law fulfilling obligations under article 11 of the WIPO copyright treaty␣
↪adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention␣
↪of such measures.
  When you convey a covered work, you waive any legal power to forbid circumvention of␣
↪technological measures to the extent such circumvention is effected by exercising␣
↪rights under this License with respect to the covered work, and you disclaim any␣
↪intention to limit operation or modification of the work as a means of enforcing,␣
↪against the work&amp;#39;s users, your or third parties&amp;#39; legal rights to forbid␣
↪circumvention of technological measures.
  4. Conveying Verbatim Copies.
  You may convey verbatim copies of the Program&amp;#39;s source code as you receive it, in␣
↪any medium, provided that you conspicuously and appropriately publish on each copy an␣
↪appropriate copyright notice; keep intact all notices stating that this License and␣
↪any non-permissive terms added in accord with section 7 apply to the code; keep intact␣
↪all notices of the absence of any warranty; and give all recipients a copy of this␣
↪License along with the Program.
  You may charge any price or no price for each copy that you convey, and you may offer␣
↪support or warranty protection for a fee.
  5. Conveying Modified Source Versions.
  You may convey a work based on the Program, or the modifications to produce it from␣
↪the Program, in the form of source code under the terms of section 4, provided that␣
↪you also meet all of these conditions:
    a) The work must carry prominent notices stating that you modified it, and giving␣
↪a relevant date.
    b) The work must carry prominent notices stating that it is released under this␣
↪License and any conditions added under section 7. This requirement modifies the␣
↪requirement in section 4 to &amp;quot;keep intact all notices&amp;quot;.
    c) You must license the entire work, as a whole, under this License to anyone who␣
↪comes into possession of a copy. This License will therefore apply, along with any␣
↪applicable section 7 additional terms, to the whole of the work, and all its parts,␣
↪regardless of how they are packaged. This License gives no permission to license the␣
↪work in any other way, but it does not invalidate such permission if you have␣
↪separately received it.
    d) If the work has interactive user interfaces, each must display Appropriate␣
↪Legal Notices; however, if the Program has interactive interfaces that do not display␣
```

(continues on next page)

(continued from previous page)

```
↪Appropriate Legal Notices, your work need not make them do so.
  A compilation of a covered work with other separate and independent works, which are␣
↪not by their nature extensions of the covered work, and which are not combined with it␣
↪such as to form a larger program, in or on a volume of a storage or distribution␣
↪medium, is called an &quot;aggregate&quot; if the compilation and its resulting␣
↪copyright are not used to limit the access or legal rights of the compilation&#39;s␣
↪users beyond what the individual works permit. Inclusion of a covered work in an␣
↪aggregate does not cause this License to apply to the other parts of the aggregate.
  6. Conveying Non-Source Forms.
  You may convey a covered work in object code form under the terms of sections 4 and 5,
↪ provided that you also convey the machine-readable Corresponding Source under the␣
↪terms of this License, in one of these ways:
    a) Convey the object code in, or embodied in, a physical product (including a␣
↪physical distribution medium), accompanied by the Corresponding Source fixed on a␣
↪durable physical medium customarily used for software interchange.
    b) Convey the object code in, or embodied in, a physical product (including a␣
↪physical distribution medium), accompanied by a written offer, valid for at least␣
↪three years and valid for as long as you offer spare parts or customer support for␣
↪that product model, to give anyone who possesses the object code either (1) a copy of␣
↪the Corresponding Source for all the software in the product that is covered by this␣
↪License, on a durable physical medium customarily used for software interchange, for a␣
↪price no more than your reasonable cost of physically performing this conveying of␣
↪source, or (2) access to copy the Corresponding Source from a network server at no␣
↪charge.
    c) Convey individual copies of the object code with a copy of the written offer to␣
↪provide the Corresponding Source. This alternative is allowed only occasionally and␣
↪noncommercially, and only if you received the object code with such an offer, in␣
↪accord with subsection 6b.
    d) Convey the object code by offering access from a designated place (gratis or␣
↪for a charge), and offer equivalent access to the Corresponding Source in the same way␣
↪through the same place at no further charge. You need not require recipients to copy␣
↪the Corresponding Source along with the object code. If the place to copy the object␣
↪code is a network server, the Corresponding Source may be on a different server␣
↪(operated by you or a third party) that supports equivalent copying facilities,␣
↪provided you maintain clear directions next to the object code saying where to find␣
↪the Corresponding Source. Regardless of what server hosts the Corresponding Source,␣
↪you remain obligated to ensure that it is available for as long as needed to satisfy␣
↪these requirements.
    e) Convey the object code using peer-to-peer transmission, provided you inform␣
↪other peers where the object code and Corresponding Source of the work are being␣
↪offered to the general public at no charge under subsection 6d.
  A separable portion of the object code, whose source code is excluded from the␣
↪Corresponding Source as a System Library, need not be included in conveying the object␣
↪code work.
  A &quot;User Product&quot; is either (1) a &quot;consumer product&quot;, which means␣
↪any tangible personal property which is normally used for personal, family, or␣
↪household purposes, or (2) anything designed or sold for incorporation into a dwelling.
↪ In determining whether a product is a consumer product, doubtful cases shall be␣
↪resolved in favor of coverage. For a particular product received by a particular user,␣
↪&quot;normally used&quot; refers to a typical or common use of that class of product,␣
↪regardless of the status of the particular user or of the way in which the particular␣
↪user actually uses, or expects or is expected to use, the product. A product is a␣
```

(continues on next page)

(continued from previous page)

```
↪consumer product regardless of whether the product has substantial commercial,␣
↪industrial or non-consumer uses, unless such uses represent the only significant mode␣
↪of use of the product.
  &amp;quot;Installation Information&amp;quot; for a User Product means any methods, procedures,
↪ authorization keys, or other information required to install and execute modified␣
↪versions of a covered work in that User Product from a modified version of its␣
↪Corresponding Source. The information must suffice to ensure that the continued␣
↪functioning of the modified object code is in no case prevented or interfered with␣
↪solely because modification has been made.
  If you convey an object code work under this section in, or with, or specifically for␣
↪use in, a User Product, and the conveying occurs as part of a transaction in which the␣
↪right of possession and use of the User Product is transferred to the recipient in␣
↪perpetuity or for a fixed term (regardless of how the transaction is characterized),␣
↪the Corresponding Source conveyed under this section must be accompanied by the␣
↪Installation Information. But this requirement does not apply if neither you nor any␣
↪third party retains the ability to install modified object code on the User Product␣
↪(for example, the work has been installed in ROM).
  The requirement to provide Installation Information does not include a requirement to␣
↪continue to provide support service, warranty, or updates for a work that has been␣
↪modified or installed by the recipient, or for the User Product in which it has been␣
↪modified or installed. Access to a network may be denied when the modification itself␣
↪materially and adversely affects the operation of the network or violates the rules␣
↪and protocols for communication across the network.
  Corresponding Source conveyed, and Installation Information provided, in accord with␣
↪this section must be in a format that is publicly documented (and with an␣
↪implementation available to the public in source code form), and must require no␣
↪special password or key for unpacking, reading or copying.
  7. Additional Terms.
  &amp;quot;Additional permissions&amp;quot; are terms that supplement the terms of this␣
↪License by making exceptions from one or more of its conditions. Additional␣
↪permissions that are applicable to the entire Program shall be treated as though they␣
↪were included in this License, to the extent that they are valid under applicable law.␣
↪If additional permissions apply only to part of the Program, that part may be used␣
↪separately under those permissions, but the entire Program remains governed by this␣
↪License without regard to the additional permissions.
  When you convey a copy of a covered work, you may at your option remove any␣
↪additional permissions from that copy, or from any part of it. (Additional permissions␣
↪may be written to require their own removal in certain cases when you modify the work.
↪) You may place additional permissions on material, added by you to a covered work,␣
↪for which you have or can give appropriate copyright permission.
  Notwithstanding any other provision of this License, for material you add to a␣
↪covered work, you may (if authorized by the copyright holders of that material)␣
↪supplement the terms of this License with terms:
    a) Disclaiming warranty or limiting liability differently from the terms of␣
↪sections 15 and 16 of this License; or
    b) Requiring preservation of specified reasonable legal notices or author␣
↪attributions in that material or in the Appropriate Legal Notices displayed by works␣
↪containing it; or
    c) Prohibiting misrepresentation of the origin of that material, or requiring that␣
↪modified versions of such material be marked in reasonable ways as different from the␣
↪original version; or
    d) Limiting the use for publicity purposes of names of licensors or authors of the␣
```

(continues on next page)

(continued from previous page)

```
↪material; or
     e) Declining to grant rights under trademark law for use of some trade names,␣
↪trademarks, or service marks; or
     f) Requiring indemnification of licensors and authors of that material by anyone␣
↪who conveys the material (or modified versions of it) with contractual assumptions of␣
↪liability to the recipient, for any liability that these contractual assumptions␣
↪directly impose on those licensors and authors.
   All other non-permissive additional terms are considered &quot;further restrictions&
↪quot; within the meaning of section 10. If the Program as you received it, or any part␣
↪of it, contains a notice stating that it is governed by this License along with a term␣
↪that is a further restriction, you may remove that term. If a license document␣
↪contains a further restriction but permits relicensing or conveying under this License,
↪ you may add to a covered work material governed by the terms of that license document,
↪ provided that the further restriction does not survive such relicensing or conveying.
   If you add terms to a covered work in accord with this section, you must place, in␣
↪the relevant source files, a statement of the additional terms that apply to those␣
↪files, or a notice indicating where to find the applicable terms.
   Additional terms, permissive or non-permissive, may be stated in the form of a␣
↪separately written license, or stated as exceptions; the above requirements apply␣
↪either way.
   8. Termination.
   You may not propagate or modify a covered work except as expressly provided under␣
↪this License. Any attempt otherwise to propagate or modify it is void, and will␣
↪automatically terminate your rights under this License (including any patent licenses␣
↪granted under the third paragraph of section 11).
   However, if you cease all violation of this License, then your license from a␣
↪particular copyright holder is reinstated (a) provisionally, unless and until the␣
↪copyright holder explicitly and finally terminates your license, and (b) permanently,␣
↪if the copyright holder fails to notify you of the violation by some reasonable means␣
↪prior to 60 days after the cessation.
   Moreover, your license from a particular copyright holder is reinstated permanently␣
↪if the copyright holder notifies you of the violation by some reasonable means, this␣
↪is the first time you have received notice of violation of this License (for any work)␣
↪from that copyright holder, and you cure the violation prior to 30 days after your␣
↪receipt of the notice.
   Termination of your rights under this section does not terminate the licenses of␣
↪parties who have received copies or rights from you under this License. If your rights␣
↪have been terminated and not permanently reinstated, you do not qualify to receive new␣
↪licenses for the same material under section 10.
   9. Acceptance Not Required for Having Copies.
   You are not required to accept this License in order to receive or run a copy of the␣
↪Program. Ancillary propagation of a covered work occurring solely as a consequence of␣
↪using peer-to-peer transmission to receive a copy likewise does not require acceptance.
↪ However, nothing other than this License grants you permission to propagate or modify␣
↪any covered work. These actions infringe copyright if you do not accept this License.␣
↪Therefore, by modifying or propagating a covered work, you indicate your acceptance of␣
↪this License to do so.
   10. Automatic Licensing of Downstream Recipients.
   Each time you convey a covered work, the recipient automatically receives a license␣
↪from the original licensors, to run, modify and propagate that work, subject to this␣
↪License. You are not responsible for enforcing compliance by third parties with this␣
↪License.
```

(continues on next page)

(continued from previous page)

```
  An &quot;entity transaction&quot; is a transaction transferring control of an␣
↪organization, or substantially all assets of one, or subdividing an organization, or␣
↪merging organizations. If propagation of a covered work results from an entity␣
↪transaction, each party to that transaction who receives a copy of the work also␣
↪receives whatever licenses to the work the party&#39;s predecessor in interest had or␣
↪could give under the previous paragraph, plus a right to possession of the␣
↪Corresponding Source of the work from the predecessor in interest, if the predecessor␣
↪has it or can get it with reasonable efforts.
  You may not impose any further restrictions on the exercise of the rights granted or␣
↪affirmed under this License. For example, you may not impose a license fee, royalty,␣
↪or other charge for exercise of rights granted under this License, and you may not␣
↪initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging␣
↪that any patent claim is infringed by making, using, selling, offering for sale, or␣
↪importing the Program or any portion of it.
  11. Patents.
  A &quot;contributor&quot; is a copyright holder who authorizes use under this License␣
↪of the Program or a work on which the Program is based. The work thus licensed is␣
↪called the contributor&#39;s &quot;contributor version&quot;.
  A contributor&#39;s &quot;essential patent claims&quot; are all patent claims owned␣
↪or controlled by the contributor, whether already acquired or hereafter acquired, that␣
↪would be infringed by some manner, permitted by this License, of making, using, or␣
↪selling its contributor version, but do not include claims that would be infringed␣
↪only as a consequence of further modification of the contributor version. For purposes␣
↪of this definition, &quot;control&quot; includes the right to grant patent sublicenses␣
↪in a manner consistent with the requirements of this License.
  Each contributor grants you a non-exclusive, worldwide, royalty-free patent license␣
↪under the contributor&#39;s essential patent claims, to make, use, sell, offer for␣
↪sale, import and otherwise run, modify and propagate the contents of its contributor␣
↪version.
  In the following three paragraphs, a &quot;patent license&quot; is any express␣
↪agreement or commitment, however denominated, not to enforce a patent (such as an␣
↪express permission to practice a patent or covenant not to s ue for patent␣
↪infringement). To &quot;grant&quot; such a patent license to a party means to make␣
↪such an agreement or commitment not to enforce a patent against the party.
  If you convey a covered work, knowingly relying on a patent license, and the␣
↪Corresponding Source of the work is not available for anyone to copy, free of charge␣
↪and under the terms of this License, through a publicly available network server or␣
↪other readily accessible means, then you must either (1) cause the Corresponding␣
↪Source to be so available, or (2) arrange to deprive yourself of the benefit of the␣
↪patent license for this particular work, or (3) arrange, in a manner consistent with␣
↪the requirements of this License, to extend the patent
  license to downstream recipients. &quot;Knowingly relying&quot; means you have actual␣
↪knowledge that, but for the patent license, your conveying the covered work in a␣
↪country, or your recipient&#39;s use of the covered work in a country, would infringe␣
↪one or more identifiable patents in that country that you have reason to believe are␣
↪valid.
  If, pursuant to or in connection with a single transaction or arrangement, you convey,
↪ or propagate by procuring conveyance of, a covered work, and grant a patent license␣
↪to some of the parties receiving the covered work authorizing them to use, propagate,␣
↪modify or convey a specific copy of the covered work, then the patent license you␣
↪grant is automatically extended to all recipients of the covered work and works based␣
↪on it.
```

(continues on next page)

(continued from previous page)

```
  A patent license is &quot;discriminatory&quot; if it does not include within the␣
↪scope of its coverage, prohibits the exercise of, or is conditioned on the non-
↪exercise of one or more of the rights that are specifically granted under this License.
↪ You may not convey a covered work if you are a party to an arrangement with a third␣
↪party that is in the business of distributing software, under which you make payment␣
↪to the third party based on the extent of your activity of conveying the work, and␣
↪under which the third party grants, to any of the parties who would receive the␣
↪covered work from you, a discriminatory patent license (a) in connection with copies␣
↪of the covered work conveyed by you (or copies made from those copies), or (b)␣
↪primarily for and in connection with specific products or compilations that contain␣
↪the covered work, unless you entered into that arrangement, or that patent license was␣
↪granted, prior to 28 March 2007.
  Nothing in this License shall be construed as excluding or limiting any implied␣
↪license or other defenses to infringement that may otherwise be available to you under␣
↪applicable patent law.
  12. No Surrender of Others&#39; Freedom.
  If conditions are imposed on you (whether by court order, agreement or otherwise)␣
↪that contradict the conditions of this License, they do not excuse you from the␣
↪conditions of this License. If you cannot convey a covered work so as to satisfy␣
↪simultaneously your obligations under this License and any other pertinent obligations,
↪ then as a consequence you may
  not convey it at all. For example, if you agree to terms that obligate you to collect␣
↪a royalty for further conveying from those to whom you convey the Program, the only␣
↪way you could satisfy both those terms and this License would be to refrain entirely␣
↪from conveying the Program.
  13. Remote Network Interaction; Use with the GNU General Public License.
  Notwithstanding any other provision of this License, if you modify the Program, your␣
↪modified version must prominently offer all users interacting with it remotely through␣
↪a computer network (if your version supports such interaction) an opportunity to␣
↪receive the Corresponding Source of your version by providing access to the␣
↪Corresponding Source from a network server at no charge, through some standard or␣
↪customary means of facilitating copying of software. This Corresponding Source shall␣
↪include the Corresponding Source for any work covered by version 3 of the GNU General␣
↪Public License that is incorporated pursuant to the following paragraph.
  Notwithstanding any other provision of this License, you have permission to link or␣
↪combine any covered work with a work licensed under version 3 of the GNU General␣
↪Public License into a single combined work, and to convey the resulting work. The␣
↪terms of this License will continue to apply to the part which is the covered work,␣
↪but the work with which it is combined will remain governed by version 3 of the GNU␣
↪General Public License.
  14. Revised Versions of this License.
  The Free Software Foundation may publish revised and/or new versions of the GNU␣
↪Affero General Public License from time to time. Such new versions will be similar in␣
↪spirit to the present version, but may differ in detail to address new problems or␣
↪concerns.
  Each version is given a distinguishing version number. If the Program specifies that␣
↪a certain numbered version of the GNU Affero General Public License &quot;or any later␣
↪version&quot; applies to it, you have the option of following the terms and conditions␣
↪either of that numbered version or of any later version published by the Free Software␣
↪Foundation. If the Program does not specify a version number of the GNU Affero General␣
↪Public License, you may choose any version ever published by the Free Software␣
↪Foundation.
```

(continues on next page)

(continued from previous page)

```
   If the Program specifies that a proxy can decide which future versions of the GNU␣
↪Affero General Public License can be used, that proxy&#39;s public statement of␣
↪acceptance of a version permanently authorizes you to choose that version for the␣
↪Program.
   Later license versions may give you additional or different permissions. However, no␣
↪additional obligations are imposed on any author or copyright holder as a result of␣
↪your choosing to follow a later version.
   15. Disclaimer of Warranty.
   THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.␣
↪EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES␣
↪PROVIDE THE PROGRAM &quot;AS IS&quot; WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED␣
↪OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY␣
↪AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND␣
↪PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME␣
↪THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
   16. Limitation of Liability.
   IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY␣
↪COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS␣
↪PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL,␣
↪INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE␣
↪PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE␣
↪OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE␣
↪WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE␣
↪POSSIBILITY OF SUCH DAMAGES.
   17. Interpretation of Sections 15 and 16.
   If the disclaimer of warranty and limitation of liability provided above cannot be␣
↪given local legal effect according to their terms, reviewing courts shall apply local␣
↪law that most closely approximates an absolute waiver of all civil liability in␣
↪connection with the Program, unless a warranty or assumption of liability accompanies␣
↪a copy of the Program in return for a fee. END OF TERMS AND CONDITIONS
How to Apply These Terms to Your New Programs
If you develop a new program, and you want it to be of the greatest possible use to the␣
↪public, the best way to achieve this is to make it free software which everyone can␣
↪redistribute and change under these terms.
To do so, attach the following notices to the program. It is safest to attach them to␣
↪the start of each source file to most effectively state the exclusion of warranty; and␣
↪each file should have at least the &quot;copyright&quot; line and a pointer to where␣
↪the full notice is found.
&lt;one line to give the program&#39;s name and a brief idea of what it does.&gt;
Copyright (C) &lt;year&gt; &lt;name of author&gt;
This program is free software: you can redistribute it and/or modify it under the terms␣
↪of the GNU Affero General Public License as published by the Free Software Foundation,␣
↪either version 3 of the License, or (at your option) any later version.
This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
↪ without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR␣
↪PURPOSE. See the GNU Affero General Public License for more details.
You should have received a copy of the GNU Affero General Public License along with this␣
↪program. If not, see &lt;http s ://www.gnu.org/licenses/&gt;.
Also add information on how to contact you by electronic and paper mail.
If your software can interact with users remotely through a computer network, you should␣
↪also make sure that it provides a way for users to get its source. For example, if␣
↪your program is a web application, its interface could display a &quot;Source&quot;␣
```

(continues on next page)

(continued from previous page)

```
↪link that leads users to an archive of the code. There are many ways you could offer␣
↪source, and different solutions will be better for different programs; see section 13␣
↪for the specific requirements.
You should also get your employer (if you work as a programmer) or school, if any, to␣
↪sign a &quot;copyright disclaimer&quot; for the program, if necessary. For more␣
↪information on this, and how to apply and follow the GNU AGPL, see &lt;http s ://www.
↪gnu.org/licenses/&gt;.
```

15.7 AGPL-3.0-or-later

```
GNU AFFERO GENERAL PUBLIC LICENSE
Version 3, 19 November 2007
Copyright (C) 2007 Free Software Foundation, Inc. &lt;http s ://fsf.org/&gt;
Everyone is permitted to copy and distribute verbatim copies of this license document,␣
↪but changing it is not allowed.
Preamble
The GNU Affero General Public License is a free, copyleft license for software and other␣
↪kinds of works, specifically designed to ensure cooperation with the community in the␣
↪case of network server software.
The licenses for most software and other practical works are designed to take away your␣
↪freedom to share and change the works. By contrast, our General Public Licenses are␣
↪intended to guarantee your freedom to share and change all versions of a program--to␣
↪make sure it remains free software for all its users.
When we speak of free software, we are referring to freedom, not price. Our General␣
↪Public Licenses are designed to make sure that you have the freedom to distribute␣
↪copies of free software (and charge for them if you wish), that you receive source␣
↪code or can get it if you want it, that you can change the software or use pieces of␣
↪it in new free programs, and that you know you can do these things.
Developers that use our General Public Licenses protect your rights with two steps: (1)␣
↪assert copyright on the software, and (2) offer you this License which gives you legal␣
↪permission to copy, distribute and/or modify the software.
A secondary benefit of defending all users&#39; freedom is that improvements made in␣
↪alternate versions of the program, if they receive widespread use, become available␣
↪for other developers to incorporate. Many developers of free software are heartened␣
↪and encouraged by the resulting cooperation. However, in the case of software used on␣
↪network servers, this result may fail to come about. The GNU General Public License␣
↪permits making a modified version and letting the public access it on a server without␣
↪ever releasing its source code to the public.
The GNU Affero General Public License is designed specifically to ensure that, in such␣
↪cases, the modified source code becomes available to the community. It requires the␣
↪operator of a network server to provide the source code of the modified version␣
↪running there to the users of that server. Therefore, public use of a modified version,
↪ on a publicly accessible server, gives the public access to the source code of the␣
↪modified version.
An older license, called the Affero General Public License and published by Affero, was␣
↪designed to accomplish similar goals. This is a different license, not a version of␣
↪the Affero GPL, but Affero has released a new version of the Affero GPL which permits␣
↪relicensing under this license.
The precise terms and conditions for copying, distribution and modification follow.
TERMS AND CONDITIONS
```

(continues on next page)

(continued from previous page)

```
  0. Definitions.
  &quot;This License&quot; refers to version 3 of the GNU Affero General Public License.
  &quot;Copyright&quot; also means copyright-like laws that apply to other kinds of␣
↪works, such as semiconductor masks.
  &quot;The Program&quot; refers to any copyrightable work licensed under this License.␣
↪Each licensee is addressed as &quot;you&quot;. &quot;Licensees&quot; and &quot;
↪recipients&quot; may be individuals or organizations.
  To &quot;modify&quot; a work means to copy from or adapt all or part of the work in a␣
↪fashion requiring copyright permission, other than the making of an exact copy. The␣
↪resulting work is called a &quot;modified version&quot; of the earlier work or a work &
↪quot;based on&quot; the earlier work.
  A &quot;covered work&quot; means either the unmodified Program or a work based on the␣
↪Program.
  To &quot;propagate&quot; a work means to do anything with it that, without permission,
↪ would make you directly or secondarily liable for infringement under applicable␣
↪copyright law, except executing it on a computer or modifying a private copy.␣
↪Propagation includes copying, distribution (with or without modification), making␣
↪available to the public, and in some countries other activities as well.
  To &quot;convey&quot; a work means any kind of propagation that enables other parties␣
↪to make or receive copies. Mere interaction with a user through a computer network,␣
↪with no transfer of a copy, is not conveying.
  An interactive user interface displays &quot;Appropriate Legal Notices&quot; to the␣
↪extent that it includes a convenient and prominently visible feature that (1) displays␣
↪an appropriate copyright notice, and (2) tells the user that there is no warranty for␣
↪the work (except to the extent that warranties are provided), that licensees may␣
↪convey the work under this License, and how to view a copy of this License. If the␣
↪interface presents a list of user commands or options, such as a menu, a prominent␣
↪item in the list meets this criterion.
  1. Source Code.
  The &quot;source code&quot; for a work means the preferred form of the work for␣
↪making modifications to it. &quot;Object code&quot; means any non-source form of a␣
↪work.
  A &quot;Standard Interface&quot; means an interface that either is an official␣
↪standard defined by a recognized standards body, or, in the case of interfaces␣
↪specified for a particular programming language, one that is widely used among␣
↪developers working in that language.
  The &quot;System Libraries&quot; of an executable work include anything, other than␣
↪the work as a whole, that (a) is included in the normal form of packaging a Major␣
↪Component, but which is not part of that Major Component, and (b) serves only to␣
↪enable use of the work with that Major Component, or to implement a Standard Interface␣
↪for which an implementation is available to the public in source code form. A &quot;
↪Major Component&quot;, in this context, means a major essential component (kernel,␣
↪window system, and so on) of the specific operating system (if any) on which the␣
↪executable work runs, or a compiler used to produce the work, or an object code␣
↪interpreter used to run it.
  The &quot;Corresponding Source&quot; for a work in object code form means all the␣
↪source code needed to generate, install, and (for an executable work) run the object␣
↪code and to modify the work, including scripts to control those activities. However,␣
↪it does not include the work&#39;s System Libraries, or general-purpose tools or␣
↪generally available free programs which are used unmodified in performing those␣
↪activities but which are not part of the work. For example, Corresponding Source␣
↪includes interface definition files associated with source files for the work, and the␣
```

(continues on next page)

(continued from previous page)

```
↪source code for shared libraries and dynamically linked subprograms that the work is␣
↪specifically designed to require, such as by intimate data communication or control␣
↪flow between those
   subprograms and other parts of the work.
   The Corresponding Source need not include anything that users can regenerate␣
↪automatically from other parts of the Corresponding Source.
   The Corresponding Source for a work in source code form is that same work.
   2. Basic Permissions.
   All rights granted under this License are granted for the term of copyright on the␣
↪Program, and are irrevocable provided the stated conditions are met. This License␣
↪explicitly affirms your unlimited permission to run the unmodified Program. The output␣
↪from running a covered work is covered by this License only if the output, given its␣
↪content, constitutes a covered work. This License acknowledges your rights of fair use␣
↪or other equivalent, as provided by copyright law.
   You may make, run and propagate covered works that you do not convey, without␣
↪conditions so long as your license otherwise remains in force. You may convey covered␣
↪works to others for the sole purpose of having them make modifications exclusively for␣
↪you, or provide you with facilities for running those works, provided that you comply␣
↪with the terms of this License in conveying all material for which you do not control␣
↪copyright. Those thus making or running the covered works for you must do so␣
↪exclusively on your behalf, under your direction and control, on terms that prohibit␣
↪them from making any copies of your copyrighted material outside their relationship␣
↪with you.
   Conveying under any other circumstances is permitted solely under the conditions␣
↪stated below. Sublicensing is not allowed; section 10 makes it unnecessary.
   3. Protecting Users&amp;#39; Legal Rights From Anti-Circumvention Law.
   No covered work shall be deemed part of an effective technological measure under any␣
↪applicable law fulfilling obligations under article 11 of the WIPO copyright treaty␣
↪adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention␣
↪of such measures.
   When you convey a covered work, you waive any legal power to forbid circumvention of␣
↪technological measures to the extent such circumvention is effected by exercising␣
↪rights under this License with respect to the covered work, and you disclaim any␣
↪intention to limit operation or modification of the work as a means of enforcing,␣
↪against the work&amp;#39;s users, your or third parties&amp;#39; legal rights to forbid␣
↪circumvention of technological measures.
   4. Conveying Verbatim Copies.
   You may convey verbatim copies of the Program&amp;#39;s source code as you receive it, in␣
↪any medium, provided that you conspicuously and appropriately publish on each copy an␣
↪appropriate copyright notice; keep intact all notices stating that this License and␣
↪any non-permissive terms added in accord with section 7 apply to the code; keep intact␣
↪all notices of the absence of any warranty; and give all recipients a copy of this␣
↪License along with the Program.
   You may charge any price or no price for each copy that you convey, and you may offer␣
↪support or warranty protection for a fee.
   5. Conveying Modified Source Versions.
   You may convey a work based on the Program, or the modifications to produce it from␣
↪the Program, in the form of source code under the terms of section 4, provided that␣
↪you also meet all of these conditions:
       a) The work must carry prominent notices stating that you modified it, and giving␣
↪a relevant date.
       b) The work must carry prominent notices stating that it is released under this␣
```

(continues on next page)

(continued from previous page)

```
↪License and any conditions added under section 7. This requirement modifies the␣
↪requirement in section 4 to &quot;keep intact all notices&quot;.
     c) You must license the entire work, as a whole, under this License to anyone who␣
↪comes into possession of a copy. This License will therefore apply, along with any␣
↪applicable section 7 additional terms, to the whole of the work, and all its parts,␣
↪regardless of how they are packaged. This License gives no permission to license the␣
↪work in any other way, but it does not invalidate such permission if you have␣
↪separately received it.
     d) If the work has interactive user interfaces, each must display Appropriate␣
↪Legal Notices; however, if the Program has interactive interfaces that do not display␣
↪Appropriate Legal Notices, your work need not make them do so.
  A compilation of a covered work with other separate and independent works, which are␣
↪not by their nature extensions of the covered work, and which are not combined with it␣
↪such as to form a larger program, in or on a volume of a storage or distribution␣
↪medium, is called an &quot;aggregate&quot; if the compilation and its resulting␣
↪copyright are not used to limit the access or legal rights of the compilation&#39;s␣
↪users beyond what the individual works permit. Inclusion of a covered work in an␣
↪aggregate does not cause this License to apply to the other parts of the aggregate.
  6. Conveying Non-Source Forms.
  You may convey a covered work in object code form under the terms of sections 4 and 5,
↪ provided that you also convey the machine-readable Corresponding Source under the␣
↪terms of this License, in one of these ways:
     a) Convey the object code in, or embodied in, a physical product (including a␣
↪physical distribution medium), accompanied by the Corresponding Source fixed on a␣
↪durable physical medium customarily used for software interchange.
     b) Convey the object code in, or embodied in, a physical product (including a␣
↪physical distribution medium), accompanied by a written offer, valid for at least␣
↪three years and valid for as long as you offer spare parts or customer support for␣
↪that product model, to give anyone who possesses the object code either (1) a copy of␣
↪the Corresponding Source for all the software in the product that is covered by this␣
↪License, on a durable physical medium customarily used for software interchange, for a␣
↪price no more than your reasonable cost of physically performing this conveying of␣
↪source, or (2) access to copy the Corresponding Source from a network server at no␣
↪charge.
     c) Convey individual copies of the object code with a copy of the written offer to␣
↪provide the Corresponding Source. This alternative is allowed only occasionally and␣
↪noncommercially, and only if you received the object code with such an offer, in␣
↪accord with subsection 6b.
     d) Convey the object code by offering access from a designated place (gratis or␣
↪for a charge), and offer equivalent access to the Corresponding Source in the same way␣
↪through the same place at no further charge. You need not require recipients to copy␣
↪the Corresponding Source along with the object code. If the place to copy the object␣
↪code is a network server, the Corresponding Source may be on a different server␣
↪(operated by you or a third party) that supports equivalent copying facilities,␣
↪provided you maintain clear directions next to the object code saying where to find␣
↪the Corresponding Source. Regardless of what server hosts the Corresponding Source,␣
↪you remain obligated to ensure that it is available for as long as needed to satisfy␣
↪these requirements.
     e) Convey the object code using peer-to-peer transmission, provided you inform␣
↪other peers where the object code and Corresponding Source of the work are being␣
↪offered to the general public at no charge under subsection 6d.
  A separable portion of the object code, whose source code is excluded from the␣
```

(continues on next page)

(continued from previous page)

```
↪Corresponding Source as a System Library, need not be included in conveying the object␣
↪code work.
  A &quot;User Product&quot; is either (1) a &quot;consumer product&quot;, which means␣
↪any tangible personal property which is normally used for personal, family, or␣
↪household purposes, or (2) anything designed or sold for incorporation into a dwelling.
↪ In determining whether a product is a consumer product, doubtful cases shall be␣
↪resolved in favor of coverage. For a particular product received by a particular user,␣
↪&quot;normally used&quot; refers to a typical or common use of that class of product,␣
↪regardless of the status of the particular user or of the way in which the particular␣
↪user actually uses, or expects or is expected to use, the product. A product is a␣
↪consumer product regardless of whether the product has substantial commercial,␣
↪industrial or non-consumer uses, unless such uses represent the only significant mode␣
↪of use of the product.
  &quot;Installation Information&quot; for a User Product means any methods, procedures,
↪ authorization keys, or other information required to install and execute modified␣
↪versions of a covered work in that User Product from a modified version of its␣
↪Corresponding Source. The information must suffice to ensure that the continued␣
↪functioning of the modified object code is in no case prevented or interfered with␣
↪solely because modification has been made.
  If you convey an object code work under this section in, or with, or specifically for␣
↪use in, a User Product, and the conveying occurs as part of a transaction in which the␣
↪right of possession and use of the User Product is transferred to the recipient in␣
↪perpetuity or for a fixed term (regardless of how the transaction is characterized),␣
↪the Corresponding Source conveyed under this section must be accompanied by the␣
↪Installation Information. But this requirement does not apply if neither you nor any␣
↪third party retains the ability to install modified object code on the User Product␣
↪(for example, the work has been installed in ROM).
  The requirement to provide Installation Information does not include a requirement to␣
↪continue to provide support service, warranty, or updates for a work that has been␣
↪modified or installed by the recipient, or for the User Product in which it has been␣
↪modified or installed. Access to a network may be denied when the modification itself␣
↪materially and adversely affects the operation of the network or violates the rules␣
↪and protocols for communication across the network.
  Corresponding Source conveyed, and Installation Information provided, in accord with␣
↪this section must be in a format that is publicly documented (and with an␣
↪implementation available to the public in source code form), and must require no␣
↪special password or key for unpacking, reading or copying.
  7. Additional Terms.
  &quot;Additional permissions&quot; are terms that supplement the terms of this␣
↪License by making exceptions from one or more of its conditions. Additional␣
↪permissions that are applicable to the entire Program shall be treated as though they␣
↪were included in this License, to the extent that they are valid under applicable law.␣
↪If additional permissions apply only to part of the Program, that part may be used␣
↪separately under those permissions, but the entire Program remains governed by this␣
↪License without regard to the additional permissions.
  When you convey a copy of a covered work, you may at your option remove any␣
↪additional permissions from that copy, or from any part of it. (Additional permissions␣
↪may be written to require their own removal in certain cases when you modify the work.
↪) You may place additional permissions on material, added by you to a covered work,␣
↪for which you have or can give appropriate copyright permission.
  Notwithstanding any other provision of this License, for material you add to a␣
↪covered work, you may (if authorized by the copyright holders of that material)␣
```

(continues on next page)

(continued from previous page)

```
↪supplement the terms of this License with terms:
     a) Disclaiming warranty or limiting liability differently from the terms of␣
↪sections 15 and 16 of this License; or
     b) Requiring preservation of specified reasonable legal notices or author␣
↪attributions in that material or in the Appropriate Legal Notices displayed by works␣
↪containing it; or
     c) Prohibiting misrepresentation of the origin of that material, or requiring that␣
↪modified versions of such material be marked in reasonable ways as different from the␣
↪original version; or
     d) Limiting the use for publicity purposes of names of licensors or authors of the␣
↪material; or
     e) Declining to grant rights under trademark law for use of some trade names,␣
↪trademarks, or service marks; or
     f) Requiring indemnification of licensors and authors of that material by anyone␣
↪who conveys the material (or modified versions of it) with contractual assumptions of␣
↪liability to the recipient, for any liability that these contractual assumptions␣
↪directly impose on those licensors and authors.
   All other non-permissive additional terms are considered &quot;further restrictions&
↪quot; within the meaning of section 10. If the Program as you received it, or any part␣
↪of it, contains a notice stating that it is governed by this License along with a term␣
↪that is a further restriction, you may remove that term. If a license document␣
↪contains a further restriction but permits relicensing or conveying under this License,
↪ you may add to a covered work material governed by the terms of that license document,
↪ provided that the further restriction does not survive such relicensing or conveying.
   If you add terms to a covered work in accord with this section, you must place, in␣
↪the relevant source files, a statement of the additional terms that apply to those␣
↪files, or a notice indicating where to find the applicable terms.
   Additional terms, permissive or non-permissive, may be stated in the form of a␣
↪separately written license, or stated as exceptions; the above requirements apply␣
↪either way.
   8. Termination.
   You may not propagate or modify a covered work except as expressly provided under␣
↪this License. Any attempt otherwise to propagate or modify it is void, and will␣
↪automatically terminate your rights under this License (including any patent licenses␣
↪granted under the third paragraph of section 11).
   However, if you cease all violation of this License, then your license from a␣
↪particular copyright holder is reinstated (a) provisionally, unless and until the␣
↪copyright holder explicitly and finally terminates your license, and (b) permanently,␣
↪if the copyright holder fails to notify you of the violation by some reasonable means␣
↪prior to 60 days after the cessation.
   Moreover, your license from a particular copyright holder is reinstated permanently␣
↪if the copyright holder notifies you of the violation by some reasonable means, this␣
↪is the first time you have received notice of violation of this License (for any work)␣
↪from that copyright holder, and you cure the violation prior to 30 days after your␣
↪receipt of the notice.
   Termination of your rights under this section does not terminate the licenses of␣
↪parties who have received copies or rights from you under this License. If your rights␣
↪have been terminated and not permanently reinstated, you do not qualify to receive new␣
↪licenses for the same material under section 10.
   9. Acceptance Not Required for Having Copies.
   You are not required to accept this License in order to receive or run a copy of the␣
↪Program. Ancillary propagation of a covered work occurring solely as a consequence of␣
```

(continues on next page)

(continued from previous page)

```
↪using peer-to-peer transmission to receive a copy likewise does not require acceptance.␣
↪ However, nothing other than this License grants you permission to propagate or modify␣
↪any covered work. These actions infringe copyright if you do not accept this License.␣
↪Therefore, by modifying or propagating a covered work, you indicate your acceptance of␣
↪this License to do so.
  10. Automatic Licensing of Downstream Recipients.
  Each time you convey a covered work, the recipient automatically receives a license␣
↪from the original licensors, to run, modify and propagate that work, subject to this␣
↪License. You are not responsible for enforcing compliance by third parties with this␣
↪License.
  An &quot;entity transaction&quot; is a transaction transferring control of an␣
↪organization, or substantially all assets of one, or subdividing an organization, or␣
↪merging organizations. If propagation of a covered work results from an entity␣
↪transaction, each party to that transaction who receives a copy of the work also␣
↪receives whatever licenses to the work the party&#39;s predecessor in interest had or␣
↪could give under the previous paragraph, plus a right to possession of the␣
↪Corresponding Source of the work from the predecessor in interest, if the predecessor␣
↪has it or can get it with reasonable efforts.
  You may not impose any further restrictions on the exercise of the rights granted or␣
↪affirmed under this License. For example, you may not impose a license fee, royalty,␣
↪or other charge for exercise of rights granted under this License, and you may not␣
↪initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging␣
↪that any patent claim is infringed by making, using, selling, offering for sale, or␣
↪importing the Program or any portion of it.
  11. Patents.
  A &quot;contributor&quot; is a copyright holder who authorizes use under this License␣
↪of the Program or a work on which the Program is based. The work thus licensed is␣
↪called the contributor&#39;s &quot;contributor version&quot;.
  A contributor&#39;s &quot;essential patent claims&quot; are all patent claims owned␣
↪or controlled by the contributor, whether already acquired or hereafter acquired, that␣
↪would be infringed by some manner, permitted by this License, of making, using, or␣
↪selling its contributor version, but do not include claims that would be infringed␣
↪only as a consequence of further modification of the contributor version. For purposes␣
↪of this definition, &quot;control&quot; includes the right to grant patent sublicenses␣
↪in a manner consistent with the requirements of this License.
  Each contributor grants you a non-exclusive, worldwide, royalty-free patent license␣
↪under the contributor&#39;s essential patent claims, to make, use, sell, offer for␣
↪sale, import and otherwise run, modify and propagate the contents of its contributor␣
↪version.
  In the following three paragraphs, a &quot;patent license&quot; is any express␣
↪agreement or commitment, however denominated, not to enforce a patent (such as an␣
↪express permission to practice a patent or covenant not to s ue for patent␣
↪infringement). To &quot;grant&quot; such a patent license to a party means to make␣
↪such an agreement or commitment not to enforce a patent against the party.
  If you convey a covered work, knowingly relying on a patent license, and the␣
↪Corresponding Source of the work is not available for anyone to copy, free of charge␣
↪and under the terms of this License, through a publicly available network server or␣
↪other readily accessible means, then you must either (1) cause the Corresponding␣
↪Source to be so available, or (2) arrange to deprive yourself of the benefit of the␣
↪patent license for this particular work, or (3) arrange, in a manner consistent with␣
↪the requirements of this License, to extend the patent
  license to downstream recipients. &quot;Knowingly relying&quot; means you have actual␣
```

(continues on next page)

(continued from previous page)

```
↪knowledge that, but for the patent license, your conveying the covered work in a␣
↪country, or your recipient&#39;s use of the covered work in a country, would infringe␣
↪one or more identifiable patents in that country that you have reason to believe are␣
↪valid.
  If, pursuant to or in connection with a single transaction or arrangement, you convey,
↪ or propagate by procuring conveyance of, a covered work, and grant a patent license␣
↪to some of the parties receiving the covered work authorizing them to use, propagate,␣
↪modify or convey a specific copy of the covered work, then the patent license you␣
↪grant is automatically extended to all recipients of the covered work and works based␣
↪on it.
  A patent license is &quot;discriminatory&quot; if it does not include within the␣
↪scope of its coverage, prohibits the exercise of, or is conditioned on the non-
↪exercise of one or more of the rights that are specifically granted under this License.
↪ You may not convey a covered work if you are a party to an arrangement with a third␣
↪party that is in the business of distributing software, under which you make payment␣
↪to the third party based on the extent of your activity of conveying the work, and␣
↪under which the third party grants, to any of the parties who would receive the␣
↪covered work from you, a discriminatory patent license (a) in connection with copies␣
↪of the covered work conveyed by you (or copies made from those copies), or (b)␣
↪primarily for and in connection with specific products or compilations that contain␣
↪the covered work, unless you entered into that arrangement, or that patent license was␣
↪granted, prior to 28 March 2007.
  Nothing in this License shall be construed as excluding or limiting any implied␣
↪license or other defenses to infringement that may otherwise be available to you under␣
↪applicable patent law.
  12. No Surrender of Others&#39; Freedom.
  If conditions are imposed on you (whether by court order, agreement or otherwise)␣
↪that contradict the conditions of this License, they do not excuse you from the␣
↪conditions of this License. If you cannot convey a covered work so as to satisfy␣
↪simultaneously your obligations under this License and any other pertinent obligations,
↪ then as a consequence you may
  not convey it at all. For example, if you agree to terms that obligate you to collect␣
↪a royalty for further conveying from those to whom you convey the Program, the only␣
↪way you could satisfy both those terms and this License would be to refrain entirely␣
↪from conveying the Program.
  13. Remote Network Interaction; Use with the GNU General Public License.
  Notwithstanding any other provision of this License, if you modify the Program, your␣
↪modified version must prominently offer all users interacting with it remotely through␣
↪a computer network (if your version supports such interaction) an opportunity to␣
↪receive the Corresponding Source of your version by providing access to the␣
↪Corresponding Source from a network server at no charge, through some standard or␣
↪customary means of facilitating copying of software. This Corresponding Source shall␣
↪include the Corresponding Source for any work covered by version 3 of the GNU General␣
↪Public License that is incorporated pursuant to the following paragraph.
  Notwithstanding any other provision of this License, you have permission to link or␣
↪combine any covered work with a work licensed under version 3 of the GNU General␣
↪Public License into a single combined work, and to convey the resulting work. The␣
↪terms of this License will continue to apply to the part which is the covered work,␣
↪but the work with which it is combined will remain governed by version 3 of the GNU␣
↪General Public License.
  14. Revised Versions of this License.
  The Free Software Foundation may publish revised and/or new versions of the GNU␣
```

(continues on next page)

(continued from previous page)

```
↪Affero General Public License from time to time. Such new versions will be similar in␣
↪spirit to the present version, but may differ in detail to address new problems or␣
↪concerns.
   Each version is given a distinguishing version number. If the Program specifies that␣
↪a certain numbered version of the GNU Affero General Public License &quot;or any later␣
↪version&quot; applies to it, you have the option of following the terms and conditions␣
↪either of that numbered version or of any later version published by the Free Software␣
↪Foundation. If the Program does not specify a version number of the GNU Affero General␣
↪Public License, you may choose any version ever published by the Free Software␣
↪Foundation.
   If the Program specifies that a proxy can decide which future versions of the GNU␣
↪Affero General Public License can be used, that proxy&#39;s public statement of␣
↪acceptance of a version permanently authorizes you to choose that version for the␣
↪Program.
   Later license versions may give you additional or different permissions. However, no␣
↪additional obligations are imposed on any author or copyright holder as a result of␣
↪your choosing to follow a later version.
   15. Disclaimer of Warranty.
   THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.␣
↪EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES␣
↪PROVIDE THE PROGRAM &quot;AS IS&quot; WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED␣
↪OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY␣
↪AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND␣
↪PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME␣
↪THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
   16. Limitation of Liability.
   IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY␣
↪COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS␣
↪PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL,␣
↪INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE␣
↪PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE␣
↪OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE␣
↪WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE␣
↪POSSIBILITY OF SUCH DAMAGES.
   17. Interpretation of Sections 15 and 16.
   If the disclaimer of warranty and limitation of liability provided above cannot be␣
↪given local legal effect according to their terms, reviewing courts shall apply local␣
↪law that most closely approximates an absolute waiver of all civil liability in␣
↪connection with the Program, unless a warranty or assumption of liability accompanies␣
↪a copy of the Program in return for a fee. END OF TERMS AND CONDITIONS
How to Apply These Terms to Your New Programs
If you develop a new program, and you want it to be of the greatest possible use to the␣
↪public, the best way to achieve this is to make it free software which everyone can␣
↪redistribute and change under these terms.
To do so, attach the following notices to the program. It is safest to attach them to␣
↪the start of each source file to most effectively state the exclusion of warranty; and␣
↪each file should have at least the &quot;copyright&quot; line and a pointer to where␣
↪the full notice is found.
&lt;one line to give the program&#39;s name and a brief idea of what it does.&gt;
Copyright (C) &lt;year&gt; &lt;name of author&gt;
This program is free software: you can redistribute it and/or modify it under the terms␣
↪of the GNU Affero General Public License as published by the Free Software Foundation,␣
```

(continues on next page)

(continued from previous page)

```
↪either version 3 of the License, or (at your option) any later version.
This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
↪ without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR␣
↪PURPOSE. See the GNU Affero General Public License for more details.
You should have received a copy of the GNU Affero General Public License along with this␣
↪program. If not, see &lt;http s ://www.gnu.org/licenses/&gt;.
Also add information on how to contact you by electronic and paper mail.
If your software can interact with users remotely through a computer network, you should␣
↪also make sure that it provides a way for users to get its source. For example, if␣
↪your program is a web application, its interface could display a &quot;Source&quot;␣
↪link that leads users to an archive of the code. There are many ways you could offer␣
↪source, and different solutions will be better for different programs; see section 13␣
↪for the specific requirements.
You should also get your employer (if you work as a programmer) or school, if any, to␣
↪sign a &quot;copyright disclaimer&quot; for the program, if necessary. For more␣
↪information on this, and how to apply and follow the GNU AGPL, see &lt;http s ://www.
↪gnu.org/licenses/&gt;.
```

15.8 Apache-2.0

```
Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

   TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

   1. Definitions.

      &quot;License&quot; shall mean the terms and conditions for use, reproduction,
      and distribution as defined by Sections 1 through 9 of this document.

      &quot;Licensor&quot; shall mean the copyright owner or entity authorized by
      the copyright owner that is granting the License.

      &quot;Legal Entity&quot; shall mean the union of the acting entity and all
      other entities that control, are controlled by, or are under common
      control with that entity. For the purposes of this definition,
      &quot;control&quot; means (i) the power, direct or indirect, to cause the
      direction or management of such entity, whether by contract or
      otherwise, or (ii) ownership of fifty percent (50%) or more of the
      outstanding shares, or (iii) beneficial ownership of such entity.

      &quot;You&quot; (or &quot;Your&quot;) shall mean an individual or Legal Entity
      exercising permissions granted by this License.

      &quot;Source&quot; form shall mean the preferred form for making modifications,
      including but not limited to software source code, documentation
      source, and configuration files.

      &quot;Object&quot; form shall mean any form resulting from mechanical
```

(continues on next page)

(continued from previous page)

```
    transformation or translation of a Source form, including but
    not limited to compiled object code, generated documentation,
    and conversions to other media types.

    &quot;Work&quot; shall mean the work of authorship, whether in Source or
    Object form, made available under the License, as indicated by a
    copyright notice that is included in or attached to the work
    (an example is provided in the Appendix below).

    &quot;Derivative Works&quot; shall mean any work, whether in Source or Object
    form, that is based on (or derived from) the Work and for which the
    editorial revisions, annotations, elaborations, or other modifications
    represent, as a whole, an original work of authorship. For the purposes
    of this License, Derivative Works shall not include works that remain
    separable from, or merely link (or bind by name) to the interfaces of,
    the Work and Derivative Works thereof.

    &quot;Contribution&quot; shall mean any work of authorship, including
    the original version of the Work and any modifications or additions
    to that Work or Derivative Works thereof, that is intentionally
    submitted to Licensor for inclusion in the Work by the copyright owner
    or by an individual or Legal Entity authorized to submit on behalf of
    the copyright owner. For the purposes of this definition, &quot;submitted&quot;
    means any form of electronic, verbal, or written communication sent
    to the Licensor or its representatives, including but not limited to
    communication on electronic mailing lists, source code control systems,
    and issue tracking systems that are managed by, or on behalf of, the
    Licensor for the purpose of discussing and improving the Work, but
    excluding communication that is conspicuously marked or otherwise
    designated in writing by the copyright owner as &quot;Not a Contribution.&quot;

    &quot;Contributor&quot; shall mean Licensor and any individual or Legal Entity
    on behalf of whom a Contribution has been received by Licensor and
    subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
    this License, each Contributor hereby grants to You a perpetual,
    worldwide, non-exclusive, no-charge, royalty-free, irrevocable
    copyright license to reproduce, prepare Derivative Works of,
    publicly display, publicly perform, sublicense, and distribute the
    Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
    this License, each Contributor hereby grants to You a perpetual,
    worldwide, non-exclusive, no-charge, royalty-free, irrevocable
    (except as stated in this section) patent license to make, have made,
    use, offer to sell, sell, import, and otherwise transfer the Work,
    where such license applies only to those patent claims licensable
    by such Contributor that are necessarily infringed by their
    Contribution(s) alone or by combination of their Contribution(s)
    with the Work to which such Contribution(s) was submitted. If You
    institute patent litigation against any entity (including a
```

(continues on next page)

(continued from previous page)

```
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
   granted to You under this License for that Work shall terminate
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
       the Derivative Works; and

   (d) If the Work includes a &quot;NOTICE&quot; text file as part of its
       distribution, then any Derivative Works that You distribute must
       include a readable copy of the attribution notices contained
       within such NOTICE file, excluding those notices that do not
       pertain to any part of the Derivative Works, in at least one
       of the following places: within a NOTICE text file distributed
       as part of the Derivative Works; within the Source form or
       documentation, if provided along with the Derivative Works; or,
       within a display generated by the Derivative Works, if and
       wherever such third-party notices normally appear. The contents
       of the NOTICE file are for informational purposes only and
       do not modify the License. You may add Your own attribution
       notices within Derivative Works that You distribute, alongside
       or as an addendum to the NOTICE text from the Work, provided
       that such additional attribution notices cannot be construed
       as modifying the License.

   You may add Your own copyright statement to Your modifications and
   may provide additional or different license terms and conditions
   for use, reproduction, or distribution of Your modifications, or
   for any such Derivative Works as a whole, provided Your use,
   reproduction, and distribution of the Work otherwise complies with
   the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
   Notwithstanding the above, nothing herein shall supersede or modify
```

(continues on next page)

(continued from previous page)

```
   the terms of any separate license agreement you may have executed
   with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
   names, trademarks, service marks, or product names of the Licensor,
   except as required for reasonable and customary use in describing the
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an &quot;AS IS&quot; BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets &quot;{}&quot;
   replaced with your own identifying information. (Don&#39;t include
   the brackets!)  The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
   same &quot;printed page&quot; as the copyright notice for easier
```

(continues on next page)

(continued from previous page)

```
    identification within third-party archives.

  Copyright 2013-2019 Nikolay Kim and Andrew Svetlov

  Licensed under the Apache License, Version 2.0 (the &quot;License&quot;);
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an &quot;AS IS&quot; BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
```

15.9 Apache-2.0

```
Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/ TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND␣
↪DISTRIBUTION
   1. Definitions.

      &quot;License&quot; shall mean the terms and conditions for use, reproduction, and␣
↪distribution as defined by Sections 1 through 9 of this document.

      &quot;Licensor&quot; shall mean the copyright owner or entity authorized by the␣
↪copyright owner that is granting the License.

      &quot;Legal Entity&quot; shall mean the union of the acting entity and all other␣
↪entities that control, are controlled by, or are under common control with that entity.
↪ For the purposes of this definition, &quot;control&quot; means (i) the power, direct␣
↪or indirect, to cause the direction or management of such entity, whether by contract␣
↪or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding␣
↪shares, or (iii) beneficial ownership of such entity.

      &quot;You&quot; (or &quot;Your&quot;) shall mean an individual or Legal Entity␣
↪exercising permissions granted by this License.

      &quot;Source&quot; form shall mean the preferred form for making modifications,␣
↪including but not limited to software source code, documentation source, and␣
↪configuration files.

      &quot;Object&quot; form shall mean any form resulting from mechanical␣
↪transformation or translation of a Source form, including but not limited to compiled␣
↪object code, generated documentation, and conversions to other media types.

      &quot;Work&quot; shall mean the work of authorship, whether in Source or Object␣
```

(continues on next page)

(continued from previous page)

```
↪form, made available under the License, as indicated by a copyright notice that is␣
↪included in or attached to the work (an example is provided in the Appendix below).

     &quot;Derivative Works&quot; shall mean any work, whether in Source or Object form,
↪ that is based on (or derived from) the Work and for which the editorial revisions,␣
↪annotations, elaborations, or other modifications represent, as a whole, an original␣
↪work of authorship. For the purposes of this License, Derivative Works shall not␣
↪include works that remain separable from, or merely link (or bind by name) to the␣
↪interfaces of, the Work and Derivative Works thereof.

     &quot;Contribution&quot; shall mean any work of authorship, including the original␣
↪version of the Work and any modifications or additions to that Work or Derivative␣
↪Works thereof, that is intentionally submitted to Licensor for inclusion in the Work␣
↪by the copyright owner or by an individual or Legal Entity authorized to submit on␣
↪behalf of the copyright owner. For the purposes of this definition, &quot;submitted&
↪quot; means any form of electronic, verbal, or written communication sent to the␣
↪Licensor or its representatives, including but not limited to communication on␣
↪electronic mailing lists, source code control systems, and issue tracking systems that␣
↪are managed by, or on behalf of, the Licensor for the purpose of discussing and␣
↪improving the Work, but excluding communication that is conspicuously marked or␣
↪otherwise designated in writing by the copyright owner as &quot;Not a Contribution.&
↪quot;

     &quot;Contributor&quot; shall mean Licensor and any individual or Legal Entity on␣
↪behalf of whom a Contribution has been received by Licensor and subsequently␣
↪incorporated within the Work.
   2. Grant of Copyright License. Subject to the terms and conditions of this License,␣
↪each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge,
↪ royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of,
↪ publicly display, publicly perform, sublicense, and distribute the Work and such␣
↪Derivative Works in Source or Object form.
   3. Grant of Patent License. Subject to the terms and conditions of this License, each␣
↪Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge,␣
↪royalty-free, irrevocable (except as stated in this section) patent license to make,␣
↪have made, use, offer to sell, sell, import, and otherwise transfer the Work, where␣
↪such license applies only to those patent claims licensable by such Contributor that␣
↪are necessarily infringed by their Contribution(s) alone or by combination of their␣
↪Contribution(s) with the Work to which such Contribution(s) was submitted. If You␣
↪institute patent litigation against any entity (including a cross-claim or␣
↪counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated␣
↪within the Work constitutes direct or contributory patent infringement, then any␣
↪patent licenses granted to You under this License for that Work shall terminate as of␣
↪the date such litigation is filed.
   4. Redistribution. You may reproduce and distribute copies of the Work or Derivative␣
↪Works thereof in any medium, with or without modifications, and in Source or Object␣
↪form, provided that You meet the following conditions:
     (a) You must give any other recipients of the Work or Derivative Works a copy of␣
↪this License; and
     (b) You must cause any modified files to carry prominent notices stating that You␣
↪changed the files; and
     (c) You must retain, in the Source form of any Derivative Works that You␣
↪distribute, all copyright, patent, trademark, and attribution notices from the Source␣
```

(continues on next page)

(continued from previous page)

```
↪form of the Work, excluding those notices that do not pertain to any part of the␣
↪Derivative Works; and
     (d) If the Work includes a &quot;NOTICE&quot; text file as part of its␣
↪distribution, then any Derivative Works that You distribute must include a readable␣
↪copy of the attribution notices contained within such NOTICE file, excluding those␣
↪notices that do not pertain to any part of the Derivative Works, in at least one of␣
↪the following places: within a NOTICE text file distributed as part of the Derivative␣
↪Works; within the Source form or documentation, if provided along with the Derivative␣
↪Works; or, within a display generated by the Derivative Works, if and wherever such␣
↪third-party notices normally appear. The contents of the NOTICE file are for␣
↪informational purposes only and do not modify the License. You may add Your own␣
↪attribution notices within Derivative Works that You distribute, alongside or as an␣
↪addendum to the NOTICE text from the Work, provided that such additional attribution␣
↪notices cannot be construed as modifying the License.
     You may add Your own copyright statement to Your modifications and may provide␣
↪additional or different license terms and conditions for use, reproduction, or␣
↪distribution of Your modifications, or for any such Derivative Works as a whole,␣
↪provided Your use, reproduction, and distribution of the Work otherwise complies with␣
↪the conditions stated in this License.
   5. Submission of Contributions. Unless You explicitly state otherwise, any␣
↪Contribution intentionally submitted for inclusion in the Work by You to the Licensor␣
↪shall be under the terms and conditions of this License, without any additional terms␣
↪or conditions. Notwithstanding the above, nothing herein shall supersede or modify the␣
↪terms of any separate license agreement you may have executed with Licensor regarding␣
↪such Contributions.
   6. Trademarks. This License does not grant permission to use the trade names,␣
↪trademarks, service marks, or product names of the Licensor, except as required for␣
↪reasonable and customary use in describing the origin of the Work and reproducing the␣
↪content of the NOTICE file.
   7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing,␣
↪Licensor provides the Work (and each Contributor provides its Contributions) on an &
↪quot;AS IS&quot; BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express␣
↪or implied, including, without limitation, any warranties or conditions of TITLE, NON-
↪INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely␣
↪responsible for determining the appropriateness of using or redistributing the Work␣
↪and assume any risks associated with Your exercise of permissions under this License.
   8. Limitation of Liability. In no event and under no legal theory, whether in tort␣
↪(including negligence), contract, or otherwise, unless required by applicable law␣
↪(such as deliberate and grossly negligent acts) or agreed to in writing, shall any␣
↪Contributor be liable to You for damages, including any direct, indirect, special,␣
↪incidental, or consequential damages of any character arising as a result of this␣
↪License or out of the use or inability to use the Work (including but not limited to␣
↪damages for loss of goodwill, work stoppage, computer failure or malfunction, or any␣
↪and all other commercial damages or losses), even if such Contributor has been advised␣
↪of the possibility of such damages.
   9. Accepting Warranty or Additional Liability. While redistributing the Work or␣
↪Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of␣
↪support, warranty, indemnity, or other liability obligations and/or rights consistent␣
↪with this License. However, in accepting such obligations, You may act only on Your␣
↪own behalf and on Your sole responsibility, not on behalf of any other Contributor,␣
↪and only if You agree to indemnify, defend, and hold each Contributor harmless for any␣
↪liability incurred by, or claims asserted against, such Contributor by reason of your␣
```

(continues on next page)

(continued from previous page)

```
↪accepting any such warranty or additional liability. END OF TERMS AND CONDITIONS
APPENDIX: How to apply the Apache License to your work.
To apply the Apache License to your work, attach the following boilerplate notice, with␣
↪the fields enclosed by brackets &quot;[]&quot; replaced with your own identifying␣
↪information. (Don&#39;t include the brackets!) The text should be enclosed in the␣
↪appropriate comment syntax for the file format. We also recommend that a file or class␣
↪name and description of purpose be included on the same &quot;printed page&quot; as␣
↪the copyright notice for easier identification within third-party archives.
Copyright [yyyy] [name of copyright owner]
Licensed under the Apache License, Version 2.0 (the &quot;License&quot;);
you may not use this file except in compliance with the License.
You may obtain a copy of the License at
http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an &quot;AS IS&quot; BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

15.10 BSD-3-Clause

```
BSD 3-Clause License

Copyright (c) 2013-2021, Kim Davies
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this
   list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
   contributors may be used to endorse or promote products derived from
   this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS &quot;AS IS&quot;
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

(continues on next page)

(continued from previous page)

15.11 BSD-3-Clause

```
Copyright (c) &lt;year&gt; &lt;owner&gt;
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

   1. Redistributions of source code must retain the above copyright notice,
      this list of conditions and the following disclaimer.

   2. Redistributions in binary form must reproduce the above copyright notice,
      this list of conditions and the following disclaimer in the documentation
      and/or other materials provided with the distribution.

   3. Neither the name of the copyright holder nor the names of its
      contributors may be used to endorse or promote products derived from
      this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS &quot;AS IS&quot;
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

15.12 BSD-3-Clause

```
Copyright (c) 2010 Jonathan Hartley
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this
  list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.
```

(continues on next page)

(continued from previous page)

```
* Neither the name of the copyright holders, nor those of its contributors
  may be used to endorse or promote products derived from this software without
  specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS &quot;AS IS&quot; AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

15.13 BSD-3-Clause

```
Copyright 2014 Pallets

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1.  Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.

2.  Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.

3.  Neither the name of the copyright holder nor the names of its
    contributors may be used to endorse or promote products derived from
    this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
&quot;AS IS&quot; AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

15.14 Apache-2.0

```
Copyright 2016-2020 aio-libs collaboration.

Licensed under the Apache License, Version 2.0 (the &quot;License&quot;);
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

   http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an &quot;AS IS&quot; BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

15.15 MIT

```
MIT License
Copyright (c) &lt;year&gt; &lt;copyright holders&gt;

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the &quot;Software&quot;), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED &quot;AS IS&quot;, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

15.16 MIT

```
MIT License

Copyright (c) 2008-2020 Andrey Petrov and contributors.

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the &quot;Software&quot;), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED &quot;AS IS&quot;, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

15.17 MIT

```
MIT License

Copyright (c) 2017

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the &quot;Software&quot;), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED &quot;AS IS&quot;, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

15.18 MIT

```
MIT License

Copyright (c) 2019 TAHRI Ahmed R.

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the &quot;Software&quot;), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED &quot;AS IS&quot;, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

15.19 MIT

```
MIT License

Copyright (c) 2021 Taneli Hukkinen

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the &quot;Software&quot;), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED &quot;AS IS&quot;, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

15.20 MIT

```
MIT License

Copyright (c) 2023 Retro for wut?

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the &amp;quot;Software&amp;quot;), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED &amp;quot;AS IS&amp;quot;, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

15.21 MIT

```
MIT License

Copyright (c) 2019 Delgan

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the &amp;quot;Software&amp;quot;), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED &amp;quot;AS IS&amp;quot;, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

15.22 MPL-2.0

```
Mozilla Public License Version 2.0
   1. Definitions
      1.1. &quot;Contributor&quot; means each individual or legal entity that creates,␣
↪contributes to the creation of, or owns Covered Software.
      1.2. &quot;Contributor Version&quot; means the combination of the Contributions of␣
↪others (if any) used by a Contributor and that particular Contributor&#39;s␣
↪Contribution.
      1.3. &quot;Contribution&quot; means Covered Software of a particular Contributor.
      1.4. &quot;Covered Software&quot; means Source Code Form to which the initial␣
↪Contributor has attached the notice in Exhibit A, the Executable Form of such Source␣
↪Code Form, and Modifications of such Source Code Form, in each case including portions␣
↪thereof.
      1.5. &quot;Incompatible With Secondary Licenses&quot; means
         (a) that the initial Contributor has attached the notice described in Exhibit B␣
↪to the Covered Software; or
         (b) that the Covered Software was made available under the terms of version 1.1␣
↪or earlier of the License, but not also under the terms of a Secondary License.
      1.6. &quot;Executable Form&quot; means any form of the work other than Source Code␣
↪Form.
      1.7. &quot;Larger Work&quot; means a work that combines Covered Software with␣
↪other material, in a separate file or files, that is not Covered Software.
      1.8. &quot;License&quot; means this document.
      1.9. &quot;Licensable&quot; means having the right to grant, to the maximum extent␣
↪possible, whether at the time of the initial grant or subsequently, any and all of the␣
↪rights conveyed by this License.
      1.10. &quot;Modifications&quot; means any of the following:
         (a) any file in Source Code Form that results from an addition to, deletion␣
↪from, or modification of the contents of Covered Software; or
         (b) any new file in Source Code Form that contains any Covered Software.
      1.11. &quot;Patent Claims&quot; of a Contributor means any patent claim(s),␣
↪including without limitation, method, process, and apparatus claims, in any patent␣
↪Licensable by such Contributor that would be infringed, but for the grant of the␣
↪License, by the making, using, selling, offering for sale, having made, import, or␣
↪transfer of either its Contributions or its Contributor Version.
      1.12. &quot;Secondary License&quot; means either the GNU General Public License,␣
↪Version 2.0, the GNU Lesser General Public License, Version 2.1, the GNU Affero␣
↪General Public License, Version 3.0, or any later versions of those licenses.
      1.13. &quot;Source Code Form&quot; means the form of the work preferred for making␣
↪modifications.
      1.14. &quot;You&quot; (or &quot;Your&quot;) means an individual or a legal entity␣
↪exercising rights under this License. For legal entities, &quot;You&quot; includes any␣
↪entity that controls, is controlled by, or is under common control with You. For␣
↪purposes of this definition, &quot;control&quot; means (a) the power, direct or␣
↪indirect, to cause the direction or management of such entity, whether by contract or␣
↪otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares␣
↪or beneficial ownership of such entity.
   2. License Grants and Conditions
      2.1. Grants
      Each Contributor hereby grants You a world-wide, royalty-free, non-exclusive␣
↪license:
```

(continues on next page)

(continued from previous page)

```
        (a) under intellectual property rights (other than patent or trademark)␣
↪Licensable by such Contributor to use, reproduce, make available, modify, display,␣
↪perform, distribute, and otherwise exploit its Contributions, either on an unmodified␣
↪basis, with Modifications, or as part of a Larger Work; and
        (b) under Patent Claims of such Contributor to make, use, sell, offer for sale,␣
↪have made, import, and otherwise transfer either its Contributions or its Contributor␣
↪Version.
     2.2. Effective Date
     The licenses granted in Section 2.1 with respect to any Contribution become␣
↪effective for each Contribution on the date the Contributor first distributes such␣
↪Contribution.
     2.3. Limitations on Grant Scope
     The licenses granted in this Section 2 are the only rights granted under this␣
↪License. No additional rights or licenses will be implied from the distribution or␣
↪licensing of Covered Software under this License. Notwithstanding Section 2.1(b) above,
↪ no patent license is granted by a Contributor:
        (a) for any code that a Contributor has removed from Covered Software; or
        (b) for infringements caused by: (i) Your and any other third party&#39;s␣
↪modifications of Covered Software, or (ii) the combination of its Contributions with␣
↪other software (except as part of its Contributor Version); or
        (c) under Patent Claims infringed by Covered Software in the absence of its␣
↪Contributions.
     This License does not grant any rights in the trademarks, service marks, or logos␣
↪of any Contributor (except as may be necessary to comply with the notice requirements␣
↪in Section 3.4).
     2.4. Subsequent Licenses
     No Contributor makes additional grants as a result of Your choice to distribute␣
↪the Covered Software under a subsequent version of this License (see Section 10.2) or␣
↪under the terms of a Secondary License (if permitted under the terms of Section 3.3).
     2.5. Representation
     Each Contributor represents that the Contributor believes its Contributions are␣
↪its original creation(s) or it has sufficient rights to grant the rights to its␣
↪Contributions conveyed by this License.
     2.6. Fair Use
     This License is not intended to limit any rights You have under applicable␣
↪copyright doctrines of fair use, fair dealing, or other equivalents.
     2.7. Conditions
     Sections 3.1, 3.2, 3.3, and 3.4 are conditions of the licenses granted in Section␣
↪2.1.
  3. Responsibilities
     3.1. Distribution of Source Form
     All distribution of Covered Software in Source Code Form, including any␣
↪Modifications that You create or to which You contribute, must be under the terms of␣
↪this License. You must inform recipients that the Source Code Form of the Covered␣
↪Software is governed by the terms of this License, and how they can obtain a copy of␣
↪this License. You may not attempt to alter or restrict the recipients&#39; rights in␣
↪the Source Code Form.
     3.2. Distribution of Executable Form
     If You distribute Covered Software in Executable Form then:
        (a) such Covered Software must also be made available in Source Code Form, as␣
↪described in Section 3.1, and You must inform recipients of the Executable Form how␣
↪they can obtain a copy of such Source Code Form by reasonable means in a timely manner,
```

(continues on next page)

(continued from previous page)

```
↪ at a charge no more than the cost of distribution to the recipient; and
        (b) You may distribute such Executable Form under the terms of this License, or␣
↪sublicense it under different terms, provided that the license for the Executable Form␣
↪does not attempt to limit or alter the recipients&#39; rights in the Source Code Form␣
↪under this License.
     3.3. Distribution of a Larger Work
     You may create and distribute a Larger Work under terms of Your choice, provided␣
↪that You also comply with the requirements of this License for the Covered Software.␣
↪If the Larger Work is a combination of Covered Software with a work governed by one or␣
↪more Secondary Licenses, and the Covered Software is not Incompatible With Secondary␣
↪Licenses, this License permits You to additionally distribute such Covered Software␣
↪under the terms of such Secondary License(s), so that the recipient of the Larger Work␣
↪may, at their option, further distribute the Covered Software under the terms of␣
↪either this License or such Secondary License(s).
     3.4. Notices
     You may not remove or alter the substance of any license notices (including␣
↪copyright notices, patent notices, disclaimers of warranty, or limitations of␣
↪liability) contained within the Source Code Form of the Covered Software, except that␣
↪You may alter any license notices to the extent required to remedy known factual␣
↪inaccuracies.
     3.5. Application of Additional Terms
     You may choose to offer, and to charge a fee for, warranty, support, indemnity or␣
↪liability obligations to one or more recipients of Covered Software. However, You may␣
↪do so only on Your own behalf, and not on behalf of any Contributor. You must make it␣
↪absolutely clear that any such warranty, support, indemnity, or liability obligation␣
↪is offered by You alone, and You hereby agree to indemnify every Contributor for any␣
↪liability incurred by such Contributor as a result of warranty, support, indemnity or␣
↪liability terms You offer. You may include additional disclaimers of warranty and␣
↪limitations of liability specific to any jurisdiction.
   4. Inability to Comply Due to Statute or Regulation
   If it is impossible for You to comply with any of the terms of this License with␣
↪respect to some or all of the Covered Software due to statute, judicial order, or␣
↪regulation then You must: (a) comply with the terms of this License to the maximum␣
↪extent possible; and (b) describe the limitations and the code they affect. Such␣
↪description must be placed in a text file included with all distributions of the␣
↪Covered Software under this License. Except to the extent prohibited by statute or␣
↪regulation, such description must be sufficiently detailed for a recipient of ordinary␣
↪skill to be able to understand it.
   5. Termination
     5.1. The rights granted under this License will terminate automatically if You␣
↪fail to comply with any of its terms. However, if You become compliant, then the␣
↪rights granted under this License from a particular Contributor are reinstated (a)␣
↪provisionally, unless and until such Contributor explicitly and finally terminates␣
↪Your grants, and (b) on an ongoing basis, if such Contributor fails to notify You of␣
↪the non-compliance by some reasonable means prior to 60 days after You have come back␣
↪into compliance. Moreover, Your grants from a particular Contributor are reinstated on␣
↪an ongoing basis if such Contributor notifies You of the non-compliance by some␣
↪reasonable means, this is the first time You have received notice of non-compliance␣
↪with this License from such Contributor, and You become compliant prior to 30 days␣
↪after Your receipt of the notice.
     5.2. If You initiate litigation against any entity by asserting a patent␣
↪infringement claim (excluding declaratory judgment actions, counter-claims, and cross-
```

(continues on next page)

(continued from previous page)

```
↪claims) alleging that a Contributor Version directly or indirectly infringes any␣
↪patent, then the rights granted to You by any and all Contributors for the Covered␣
↪Software under Section 2.1 of this License shall terminate.
     5.3. In the event of termination under Sections 5.1 or 5.2 above, all end user␣
↪license agreements (excluding distributors and resellers) which have been validly␣
↪granted by You or Your distributors under this License prior to termination shall␣
↪survive termination.
  6. Disclaimer of Warranty
  Covered Software is provided under this License on an &quot;as is&quot; basis,␣
↪without warranty of any kind, either expressed, implied, or statutory, including,␣
↪without limitation, warranties that the Covered Software is free of defects,␣
↪merchantable, fit for a particular purpose or non-infringing. The entire risk as to␣
↪the quality and performance of the Covered Software is with You. Should any Covered␣
↪Software prove defective in any respect, You (not any Contributor) assume the cost of␣
↪any necessary servicing, repair, or correction. This disclaimer of warranty␣
↪constitutes an essential part of this License. No use of any Covered Software is␣
↪authorized under this License except under this disclaimer.
  7. Limitation of Liability
  Under no circumstances and under no legal theory, whether tort (including negligence),
↪ contract, or otherwise, shall any Contributor, or anyone who distributes Covered␣
↪Software as permitted above, be liable to You for any direct, indirect, special,␣
↪incidental, or consequential damages of any character including, without limitation,␣
↪damages for lost profits, loss of goodwill, work stoppage, computer failure or␣
↪malfunction, or any and all other commercial damages or losses, even if such party␣
↪shall have been informed of the possibility of such damages. This limitation of␣
↪liability shall not apply to liability for death or personal injury resulting from␣
↪such party&#39;s negligence to the extent applicable law prohibits such limitation.␣
↪Some jurisdictions do not allow the exclusion or limitation of incidental or␣
↪consequential damages, so this exclusion and limitation may not apply to You.
  8. Litigation
  Any litigation relating to this License may be brought only in the courts of a␣
↪jurisdiction where the defendant maintains its principal place of business and such␣
↪litigation shall be governed by laws of that jurisdiction, without reference to its␣
↪conflict-of-law provisions. Nothing in this Section shall prevent a party&#39;s␣
↪ability to bring cross-claims or counter-claims.
  9. Miscellaneous
  This License represents the complete agreement concerning the subject matter hereof.␣
↪If any provision of this License is held to be unenforceable, such provision shall be␣
↪reformed only to the extent necessary to make it enforceable. Any law or regulation␣
↪which provides that the language of a contract shall be construed against the drafter␣
↪shall not be used to construe this License against a Contributor.
  10. Versions of the License
     10.1. New Versions
     Mozilla Foundation is the license steward. Except as provided in Section 10.3, no␣
↪one other than the license steward has the right to modify or publish new versions of␣
↪this License. Each version will be given a distinguishing version number.
     10.2. Effect of New Versions
     You may distribute the Covered Software under the terms of the version of the␣
↪License under which You originally received the Covered Software, or under the terms␣
↪of any subsequent version published by the license steward.
     10.3. Modified Versions
     If you create software not governed by this License, and you want to create a new␣
```

(continues on next page)

(continued from previous page)

```
↪license for such software, you may create and use a modified version of this License␣
↪if you rename the license and remove any references to the name of the license steward␣
↪(except to note that such modified license differs from this License).
      10.4. Distributing Source Code Form that is Incompatible With Secondary Licenses
      If You choose to distribute Source Code Form that is Incompatible With Secondary␣
↪Licenses under the terms of this version of the License, the notice described in␣
↪Exhibit B of this License must be attached. Exhibit A - Source Code Form License Notice
This Source Code Form is subject to the terms of the Mozilla Public License, v. 2.0. If␣
↪a copy of the MPL was not distributed with this file, You can obtain one at http://
↪mozilla.org/MPL/2.0/.
If it is not possible or desirable to put the notice in a particular file, then You may␣
↪include the notice in a location (such as a LICENSE file in a relevant directory)␣
↪where a recipient would be likely to look for such a notice.
You may add additional accurate notices of copyright ownership.
Exhibit B - &quot;Incompatible With Secondary Licenses&quot; Notice
This Source Code Form is &quot;Incompatible With Secondary Licenses&quot;, as defined by␣
↪the Mozilla Public License, v. 2.0.
```

15.23 Python-2.0

```
PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2
   1. This LICENSE AGREEMENT is between the Python Software Foundation (&quot;PSF&quot;),
↪ and the Individual or Organization (&quot;Licensee&quot;) accessing and otherwise␣
↪using this software (&quot;Python&quot;) in source or binary form and its associated␣
↪documentation.
   2. Subject to the terms and conditions of this License Agreement, PSF hereby grants␣
↪Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test,␣
↪perform and/or display publicly, prepare derivative works, distribute, and otherwise␣
↪use Python alone or in any derivative version, provided, however, that PSF&#39;s␣
↪License Agreement and PSF&#39;s notice of copyright, i.e., &quot;Copyright (c) 2001,␣
↪2002, 2003, 2004, 2005, 2006 Python Software Foundation; All Rights Reserved&quot; are␣
↪retained in Python alone or in any derivative version prepared by Licensee.
   3. In the event Licensee prepares a derivative work that is based on or incorporates␣
↪Python or any part thereof, and wants to make the derivative work available to others␣
↪as provided herein, then Licensee hereby agrees to include in any such work a brief␣
↪summary of the changes made to Python.
   4. PSF is making Python available to Licensee on an &quot;AS IS&quot; basis. PSF␣
↪MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT␣
↪LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF␣
↪MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON WILL␣
↪NOT INFRINGE ANY THIRD PARTY RIGHTS.
   5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON FOR ANY␣
↪INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING,␣
↪DISTRIBUTING, OR OTHERWISE USING PYTHON, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF␣
↪THE POSSIBILITY THEREOF.
   6. This License Agreement will automatically terminate upon a material breach of its␣
↪terms and conditions.
   7. Nothing in this License Agreement shall be deemed to create any relationship of␣
↪agency, partnership, or joint venture between PSF and Licensee. This License Agreement␣
↪does not grant permission to use PSF trademarks or trade name in a trademark sense to␣
```

(continues on next page)

(continued from previous page)

```
↪endorse or promote products or services of Licensee, or any third party.
   8. By copying, installing or otherwise using Python, Licensee agrees to be bound by␣
↪the terms and conditions of this License Agreement. BEOPEN.COM LICENSE AGREEMENT FOR␣
↪PYTHON 2.0
BEOPEN PYTHON OPEN SOURCE LICENSE AGREEMENT VERSION 1
   1. This LICENSE AGREEMENT is between BeOpen.com (&quot;BeOpen&quot;), having an␣
↪office at 160 Saratoga Avenue, Santa Clara, CA 95051, and the Individual or␣
↪Organization (&quot;Licensee&quot;) accessing and otherwise using this software in␣
↪source or binary form and its associated documentation (&quot;the Software&quot;).
   2. Subject to the terms and conditions of this BeOpen Python License Agreement,␣
↪BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license to␣
↪reproduce, analyze, test, perform and/or display publicly, prepare derivative works,␣
↪distribute, and otherwise use the Software alone or in any derivative version,␣
↪provided, however, that the BeOpen Python License is retained in the Software, alone␣
↪or in any derivative version prepared by Licensee.
   3. BeOpen is making the Software available to Licensee on an &quot;AS IS&quot; basis.␣
↪BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE,␣
↪BUT NOT LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF␣
↪MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE␣
↪WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
   4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY␣
↪INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING, MODIFYING␣
↪OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE␣
↪POSSIBILITY THEREOF.
   5. This License Agreement will automatically terminate upon a material breach of its␣
↪terms and conditions.
   6. This License Agreement shall be governed by and interpreted in all respects by the␣
↪law of the State of California, excluding conflict of law provisions. Nothing in this␣
↪License Agreement shall be deemed to create any relationship of agency, partnership,␣
↪or joint venture between BeOpen and Licensee. This License Agreement does not grant␣
↪permission to use BeOpen trademarks or trade names in a trademark sense to endorse or␣
↪promote products or services of Licensee, or any third party. As an exception, the &
↪quot;BeOpen Python&quot; logos available at http://www.pythonlabs.com/logos.html may␣
↪be used according to the permissions granted on that web page.
   7. By copying, installing or otherwise using the software, Licensee agrees to be␣
↪bound by the terms and conditions of this License Agreement. CNRI OPEN SOURCE LICENSE␣
↪AGREEMENT (for Python 1.6b1) IMPORTANT: PLEASE READ THE FOLLOWING AGREEMENT CAREFULLY.
BY CLICKING ON &quot;ACCEPT&quot; WHERE INDICATED BELOW, OR BY COPYING, INSTALLING OR␣
↪OTHERWISE USING PYTHON 1.6, beta 1 SOFTWARE, YOU ARE DEEMED TO HAVE AGREED TO THE␣
↪TERMS AND CONDITIONS OF THIS LICENSE AGREEMENT.
   1. This LICENSE AGREEMENT is between the Corporation for National Research␣
↪Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191 (&quot;
↪CNRI&quot;), and the Individual or Organization (&quot;Licensee&quot;) accessing and␣
↪otherwise using Python 1.6, beta 1 software in source or binary form and its␣
↪associated documentation, as released at the www.python.org Internet site on August 4,␣
↪2000 (&quot;Python 1.6b1&quot;).
   2. Subject to the terms and conditions of this License Agreement, CNRI hereby grants␣
↪Licensee a non-exclusive, royalty-free, world-wide license to reproduce, analyze, test,
↪ perform and/or display publicly, prepare derivative works, distribute, and otherwise␣
↪use Python 1.6b1 alone or in any derivative version, provided, however, that CNRIs␣
↪License Agreement is retained in Python 1.6b1, alone or in any derivative version␣
↪prepared by Licensee.
```

(continues on next page)

(continued from previous page)

```
   Alternately, in lieu of CNRIs License Agreement, Licensee may substitute the␣
↪following text (omitting the quotes): &quot;Python 1.6, beta 1, is made available␣
↪subject to the terms and conditions in CNRIs License Agreement. This Agreement may be␣
↪located on the Internet using the following unique, persistent identifier (known as a␣
↪handle): 1895.22/1011. This Agreement may also be obtained from a proxy server on the␣
↪Internet using the URL:http://hdl.handle.net/1895.22/1011&quot;.
   3. In the event Licensee prepares a derivative work that is based on or incorporates␣
↪Python 1.6b1 or any part thereof, and wants to make the derivative work available to␣
↪the public as provided herein, then Licensee hereby agrees to indicate in any such␣
↪work the nature of the modifications made to Python 1.6b1.
   4. CNRI is making Python 1.6b1 available to Licensee on an &quot;AS IS&quot; basis.␣
↪CNRI MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE,␣
↪BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF␣
↪MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6b1␣
↪WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
   5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY␣
↪INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING, MODIFYING␣
↪OR DISTRIBUTING PYTHON 1.6b1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE␣
↪POSSIBILITY THEREOF.
   6. This License Agreement will automatically terminate upon a material breach of its␣
↪terms and conditions.
   7. This License Agreement shall be governed by and interpreted in all respects by the␣
↪law of the State of Virginia, excluding conflict of law provisions. Nothing in this␣
↪License Agreement shall be deemed to create any relationship of agency, partnership,␣
↪or joint venture between CNRI and Licensee. This License Agreement does not grant␣
↪permission to use CNRI trademarks or trade name in a trademark sense to endorse or␣
↪promote products or services of Licensee, or any third party.
   8. By clicking on the &quot;ACCEPT&quot; button where indicated, or by copying,␣
↪installing or otherwise using Python 1.6b1, Licensee agrees to be bound by the terms␣
↪and conditions of this License Agreement. ACCEPT CWI LICENSE AGREEMENT FOR PYTHON 0.9.
↪0 THROUGH 1.2
Copyright (c) 1991 - 1995, Stichting Mathematisch Centrum Amsterdam, The Netherlands.␣
↪All rights reserved.
Permission to use, copy, modify, and distribute this software and its documentation for␣
↪any purpose and without fee is hereby granted, provided that the above copyright␣
↪notice appear in all copies and that both that copyright notice and this permission␣
↪notice appear in supporting documentation, and that the name of Stichting Mathematisch␣
↪Centrum or CWI not be used in advertising or publicity pertaining to distribution of␣
↪the software without specific, written prior permission.
STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,␣
↪INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL␣
↪STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL␣
↪DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER␣
↪IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN␣
↪CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
```

15.24 MIT

```
The MIT License (MIT)

Copyright (c) 2015 Hynek Schlawack and the attrs contributors

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the &quot;Software&quot;), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED &quot;AS IS&quot;, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

15.25 MIT

```
The MIT License (MIT)

Copyright (c) 2017 to present Pydantic Services Inc. and individual contributors.

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the &quot;Software&quot;), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED &quot;AS IS&quot;, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

15.26 MIT

```
The MIT License (MIT)

Copyright (c) 2017, 2018, 2019, 2020, 2021, 2022 Samuel Colvin

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the &quot;Software&quot;), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED &quot;AS IS&quot;, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

15.27 MIT

```
The MIT License (MIT)

Copyright (c) 2018 Alex Grönholm

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the &quot;Software&quot;), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED &quot;AS IS&quot;, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

15.28 MIT

```
The MIT License (MIT)

Copyright (c) 2022 Alex Grönholm

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the &quot;Software&quot;), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED &quot;AS IS&quot;, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.


This project contains code copied from the Python standard library.
The following is the required license notice for those parts.

PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2
--------------------------------------------

1. This LICENSE AGREEMENT is between the Python Software Foundation
(&quot;PSF&quot;), and the Individual or Organization (&quot;Licensee&quot;) accessing␣
↪and
otherwise using this software (&quot;Python&quot;) in source or binary form and
its associated documentation.

2. Subject to the terms and conditions of this License Agreement, PSF hereby
grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce,
analyze, test, perform and/or display publicly, prepare derivative works,
distribute, and otherwise use Python alone or in any derivative version,
provided, however, that PSF&#39;s License Agreement and PSF&#39;s notice of copyright,
i.e., &quot;Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010,
2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022 Python Software␣
↪Foundation;
All Rights Reserved&quot; are retained in Python alone or in any derivative version
prepared by Licensee.

3. In the event Licensee prepares a derivative work that is based on
or incorporates Python or any part thereof, and wants to make
the derivative work available to others as provided herein, then
Licensee hereby agrees to include in any such work a brief summary of
the changes made to Python.
```

(continues on next page)

(continued from previous page)

```
4. PSF is making Python available to Licensee on an &quot;AS IS&quot;
basis.  PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR
IMPLIED.  BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND
DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS
FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON WILL NOT
INFRINGE ANY THIRD PARTY RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON
FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS
A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON,
OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material
breach of its terms and conditions.

7. Nothing in this License Agreement shall be deemed to create any
relationship of agency, partnership, or joint venture between PSF and
Licensee.  This License Agreement does not grant permission to use PSF
trademarks or trade name in a trademark sense to endorse or promote
products or services of Licensee, or any third party.

8. By copying, installing or otherwise using Python, Licensee
agrees to be bound by the terms and conditions of this License
Agreement.
```

15.29 MIT

```
The MIT License (MIT)

Copyright (c) 2022 Samuel Colvin

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the &quot;Software&quot;), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED &quot;AS IS&quot;, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

(continues on next page)

(continued from previous page)

15.30 MIT

```
The MIT License (MIT)

Copyright (c) 2022 the contributors

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the &quot;Software&quot;), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED &quot;AS IS&quot;, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

15.31 MIT

```
The MIT License (MIT)

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
&quot;Software&quot;), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED &quot;AS IS&quot;, WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
```

(continues on next page)

(continued from previous page)

```
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

15.32 MPL-2.0

```
This package contains a modified version of ca-bundle.crt:

ca-bundle.crt -- Bundle of CA Root Certificates

This is a bundle of X.509 certificates of public Certificate Authorities
(CA). These were automatically extracted from Mozilla&#39;s root certificates
file (certdata.txt).  This file can be found in the mozilla source tree:
https://hg.mozilla.org/mozilla-central/file/tip/security/nss/lib/ckfw/builtins/certdata.
↪txt
It contains the certificates in PEM format and therefore
can be directly used with curl / libcurl / php_curl, or with
an Apache+mod_ssl webserver for SSL client authentication.
Just configure this file as the SSLCACertificateFile.#

***** BEGIN LICENSE BLOCK *****
This Source Code Form is subject to the terms of the Mozilla Public License,
v. 2.0. If a copy of the MPL was not distributed with this file, You can obtain
one at http://mozilla.org/MPL/2.0/.

***** END LICENSE BLOCK *****
@(#) $RCSfile: certdata.txt,v $ $Revision: 1.80 $ $Date: 2011/11/03 15:11:58 $
```

15.33 Apache-2.0

```
This software is made available under the terms of *either* of the
licenses found in LICENSE.APACHE2 or LICENSE.MIT. Contributions to are
made under the terms of *both* these licenses.
```

PYTHON MODULE INDEX

i

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X