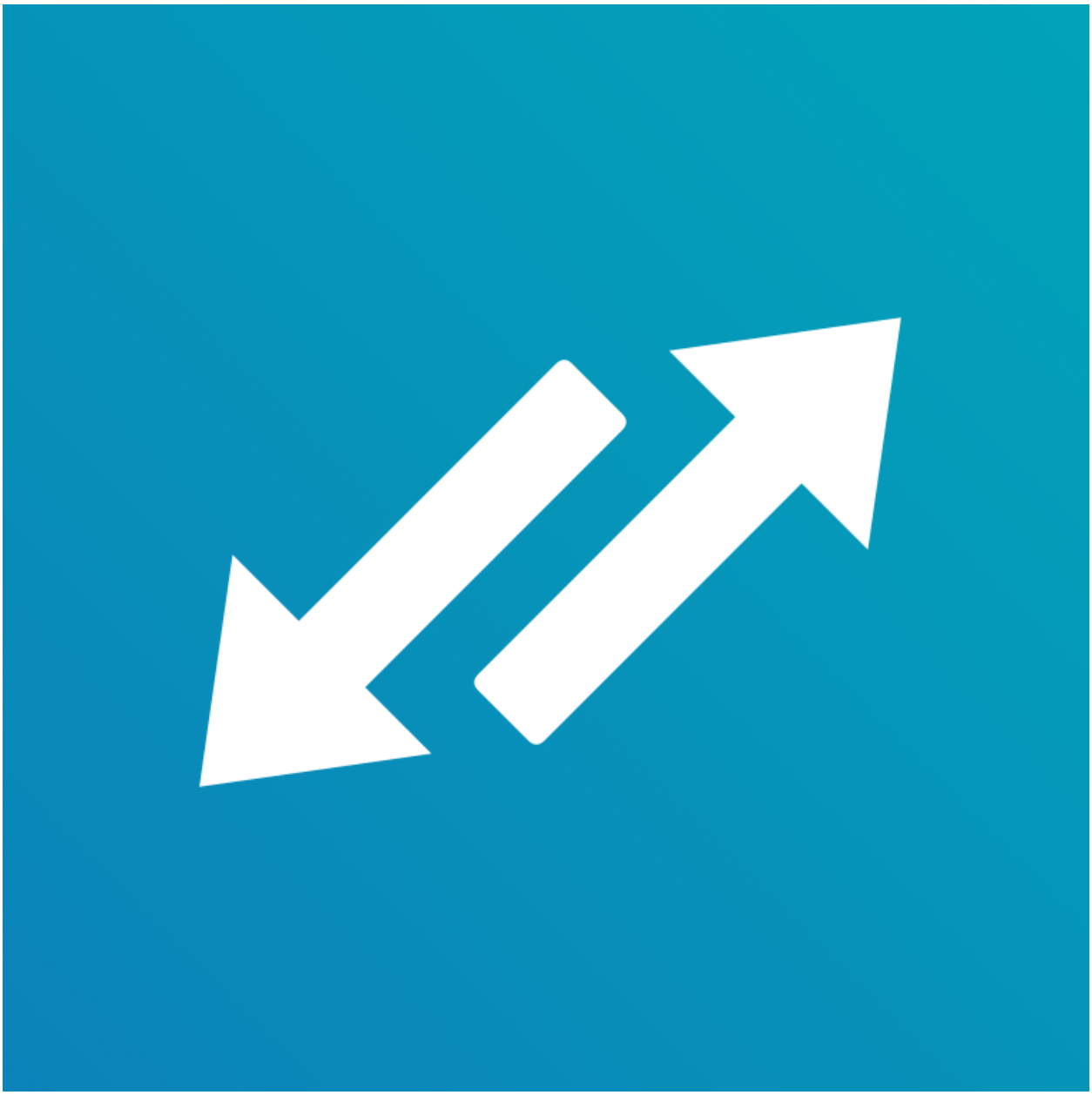


EH Forwarder Bot Documentation

Yayım 2.1.1

Eana Hufwe, and the EH Forwarder Bot contributors

04 Mar 2022

İçindekiler

EH Forwarder Bot

An extensible message tunneling chat bot framework.

Codename **EH Forwarder Bot** (EFB) is an extensible message tunneling chat bot framework which delivers messages to and from multiple platforms and remotely control your accounts.

BÖLÜM 1

Başlarken

Başlamadan önce EFB ile ilgili birkaç basit adım.

1.1 EH Forwarder Bot Yükle

EH Forwarder Bot aşağıdaki yollarla yüklenebilir:

1.1.1 PyPI'den yükle

`` pip``, varsayılan olarak PyPI'den en son kararlı sürümü kuracaktır, ancak PyPI'de geliştirme sürümleri de mevcuttur.

```
pip3 install ehforwarderbot
```

1.1.2 GitHub'dan yükle

Bu, GitHub'dan en günceli yükleme yapacaktır. Kalıcı olmayabilir, bu yüzden dikkatlice ilerleyin.

```
pip3 install git+https://github.com/ehForwarderBot/ehforwarderbot.git
```

1.1.3 Alternative installation methods

You can find a list of alternative installation methods contributed by the community in the project wiki.

For scripts, containers (e.g. Docker), etc. that may include one or more external modules, please visit the modules repository.

Not: These alternative installation methods are maintained by the community, please consult their respective author or maintainer for help related to those methods.

1.2 A stable internet connection

Since the majority of our channels are using polling for message retrieval, a stable internet connection is necessary for channels to run smoothly. An unstable connection may lead to slow response, or loss of messages.

1.3 Yerel dizinleri oluştur

EFB, *nix kullanıcı yapılandırma stilini kullanır; bu stil ayrıntılı olarak dizinler bölümünde açıklanmıştır. Kısaca, varsayılan yapılandırmayı kullanıyorsanız, `~/.ehforwarderbot` oluşturmanız ve EFB'yi çalıştıran kullanıcıya okuma ve yazma izni vermeniz gerekir.

1.4 Modülleri seç, yükle ve etkinleştir

Currently, all modules that was submitted to us are recorded in the modules repository. You can choose the channels that fits your need the best.

Her kanalın yüklenmesine ilişkin talimatlar kendi belgelerinde mevcuttur.

1.4.1 Set up with the configuration wizard

When you have successfully installed the modules of your choices, you can the use the configuration wizard which guides you to enable channels and middlewares, and continue to setup those modules if they also have provided a similar wizard.

You can start the wizard by running the following command in a compatible console or terminal emulator:

```
efb-wizard
```

If you want to start the wizard of a module for a profile individually, run:

```
efb-wizard -p <profile name> -m <module ID>
```

1.4.2 Set up manually

Alternatively, you can enable those modules manually it by listing its Channel ID in the *configuration file*. The default path is `~/.ehforwarderbot/profiles/default/config.yaml`. Please refer to *Altdizinler* if you have configured otherwise.

Aynı zamanda çalışan birden fazla bağımlı kanalınız olduğu halde, tek bir profilde çalışan tek bir ana kanalınız olabilir. Bu arada, orta sınıflar tamamen isteğe bağlıdır.

For example, to enable the following modules:

- **Ana Kanal**
  - Demo Master (`foo.demo_master`)
- **Bağımlı kanallar**
  - Demo Slave (`foo.demo_slave`)
  - Dummy Slave (`bar.dummy`)
- **Özel yazılımlar**
  - Null Middleware (`foo.null`)

`config.yaml` should have the following lines:

```
master_channel: foo.demo_master
slave_channels:
- foo.demo_slave
- bar.dummy
middlewares:
- foo.null
```

If you have enabled modules manually, you might also need configure each module manually too. Please consult the documentation of each module for instructions.

1.5 EFB'yi çalıştır

```
ehforwarderbot
```

Bu, doğrudan geçerli ortamda EFB'yi başlatacaktır. Varsayılan *Profiller* den farklı bir profile EFB'yi başlatmak için `default` olarak isimlendirilir ve komuta``–profile <profile-name>`` eklenir.

Daha fazla komut satırı seçeneği için, `--help` seçeneğini kullanın.

1.5.1 Use EFB in another language

EFB supports translated user interface and prompts. You can set your system language or locale environmental variables (`LANGUAGE`, `LC_ALL`, `LC_MESSAGES` or `LANG`) to one of our supported languages to switch language.

You can help to translate this project into your languages on our Crowdin page.

Not: If your are installing from source code, you will not get translations of the user interface without manual compile of message catalogs (`.mo`) prior to installation.

1.5.2 EFB'yi bir daemon işlemi olarak başlat

Versiyon 2'den bu yana, EH Forwarder Bot daemonun yardımcısını kullanmakta kararsız olduğu için kaldırıldı. Daemon yönetimi için, systemd, upstart yada pm2 gibi kararlı çözümler kullanmanızı öneririz.

BÖLÜM 2

Yapılandırma Dosyası

EFB has an overall configuration file to manage all enabled modules. It is located under the *directory* of current profile, and named `config.yaml`.

2.1 Syntax

The configuration file is in the YAML syntax. If you are not familiar with its syntax, please check its documentations and tutorials for details.

- The ID of the master channel enabled is under the key `master_channel`
- The ID of slave channels enabled is listed under the key `slave_channels`. It has to be a list even if just one channel is to be enabled.
- The ID of middlewares enabled are listed under the key `middlewares`. It has to be a list even if just one middleware is to be enabled. However, if you don't want to enable any middleware, just omit the section completely.

2.2 Instance ID

To have multiple accounts running simultaneously, you can appoint an instance ID to a module. Instance ID can be defined by the user, and if defined, it must has nothing other than letters, numbers and underscores, i.e. in regular expressions `[a-zA-Z0-9_]+`. When instance ID is not defined, the channel will run in the "default" instance with no instance ID.

To indicate the instance ID of an instance, append `#` following by the instance ID to the module ID. For example, slave channel `bar.dummy` running with instance ID `alice` should be written as `bar.dummy#alice`. If the channel requires configurations, it should be done in the directory with the same name (e.g. `EFB_DATA_PATH/profiles/`*`PROFILE`*`/bar.dummy#alice`), so as to isolate instances.

Please avoid having two modules with the same set of module ID and instance ID as it may leads to unexpected results.

For example, to enable the following modules:

- **Master channel**

  - Demo Master (`foo.demo_master`)

- **Slave channels**
  - Demo Slave (`foo.demo_slave`)
  - Dummy Slave (`bar.dummy`)
  - Dummy Slave (`bar.dummy`) at `alt` instance
- **Middlewares**
  - Message Archiver (`foo.msg_archiver`)
  - Null Middleware (`foo.null`)

`config.yaml` should have the following lines:

```
master_channel: foo.demo_master
slave_channels:
- foo.demo_slave
- bar.dummy
- bar.dummy#alt
middlewares:
- foo.msg_archiver
- foo.null
```

2.3 Granulated logging control

If you have special needs on processing and controlling the log produced by the framework and running modules, you can use specify the log configuration with Python's configuration dictionary schema under section `logging`.

An example of logging control settings:

```
logging:
    version: 1
    disable_existing_loggers: false
    formatters:
        standard:
            format: '%(asctime)s [%(levelname)s] %(name)s: %(message)s'
    handlers:
        default:
            level: INFO
            formatter: standard
            class: logging.StreamHandler
            stream: ext://sys.stdout
    loggers:
          '':
                handlers: [default,]
                level: INFO
                propagate: true
          AliceIRCChannel:
                handlers: [default, ]
                level: WARN
                propagate: false
```

BÖLÜM 3

Framework'ü başlat

EH Forwarder Bot, sistemi başlatmak için 2 yol önerdi:

- `ehforwarderbot`
- `python3 -m ehforwarderbot`

Her iki komut da tam olarak aynı şeydir, aynı bayrakları kabul eder, aynı kodu çalıştırır. İkincisi yalnızca ilkinin çalışmaması durumu için bir yedektir.

3.1 Seçenekler

- `-h`, `--help`: Yardım mesajını göster
- `-p` *`PROFILE`*, `--profile` *`PROFILE`*: Switch *profile*

  Sürüm 2'de EFB, profilleri tarafından tanımlanan aynı kullanıcı altındaki farklı örnekleri çalıştırmayı desteklemektedir. Varsayılan profile `default` adı verilir.

- `-V`, `--version`: Sürüm bilgilerini yazdır

  Bu, kullandığınız Python'un sürüm numarasını, EFB yapısını, tüm kanalları ve özel yazılımları etkinleştirildiğini gösterir.

- `-v`, `--verbose`: Ayrıntılı günlüğü yazdır

  This option enables verbose log of EFB and all enabled modules. This, together with `--version`, is particularly useful in debugging and issue reporting.

- `--trace-threads`: Trace hanging threads

  This option is useful to identify source of the issue when you encounter situations where you had to force quit EFB. When this option is enabled, once the first stop signal (`SIGINT` or `SIGTERM`) is sent, threads that are *asleep* will be identified and reported every 10 seconds, until a second stop signal is seen.

  In order to use this option, you need to install extra Python dependencies using the following command.

```
pip3 install 'ehforwarderbot[trace]'
```

3.2 Quitting EFB

If you started EFB in a shell, you can simply press `Control-c` to trigger the quit process. Otherwise, ask your service manager to issue a `SIGTERM` for a graceful exit. The exit process may take a few second to complete.

Önemli: It is important for you to issue a graceful termination signal (e.g. `SIGTERM`), and **NOT** to use `SIGKILL`. Otherwise you may face the risk of losing data and breaking programs.

If you have encountered any issue quitting EFB, press `Control-c` for 5 times consecutively to trigger a force quit. In case you have frequently encountered situations where you had to force quit EFB, there might be a bug with EFB or any modules enabled. You may want to use the `--trace-threads` option described above to identify the source of issue, and report this to relevant developers.

BÖLÜM 4

Altdizinler

Çoğu modüllerin EH Forwarder Bot 2.0 için konfigürasyon ve dataları "EFB data directory"de depolanırken, Python Package Manager `pip` ile yüklenmesi gerekmektedir.

Varsayılan olarak, veri dizini kullanıcıya özgü, kullanıcının ana dizininde bulunan, `` ~/.ehforwarderbot``. Bu, çevre değişkeni `EFB_DATA_PATH` ile geçersiz kılınabilir. Burada tanımlanan bu yol bir ** mutlak yol ** olmalıdır.

4.1 Altdizin yapısı

Bu kısım size EFB veri dizininin yapısını, varsayılan konfigürasyonu örnek olarak kullanarak tanıtacak.

```
./ehforwarderbot                or $EFB_DATA_PATH
|- profiles
|  |- default                   The default profile.
|  |  |- config.yaml            Main configuration file.
|  |  |- dummy_ch_master        Directory for data of the channel
|  |  |  |- config.yaml         Config file of the channel. (example)
|  |  |  |- ...
|  |  |- random_ch_slave
|  |  |  |- ...
|  |- profile2                  Alternative profile
|  |  |- config.yaml
|  |  |- ...
|  |- ...
|- modules                      Place for source code of your own channels/middlewares
|  |- random_ch_mod_slave       Channels here have a higher priority while importing
|  |  |- __init__.py
|  |  |- ...
```

BÖLÜM 5

Profiller

EFB'nin 2. sürümünden başlayarak, profiller kullanıcıların ihtiyaçları olduğunda çoklu EFB örneklerinin simültane ve birbirlerini etkilemeyecek şekilde çalışmalarına izin vermek için sunulmuştur.

Her profil kendi konfigürasyon dosyası dizisine ve aynı kodu paylaşan kanallar dizisine sahiptir, fakat farklı veri dosyaları vardır, bu yüzden kendi kendilerine çalışabilirler.

Varsayılan profil ismi şudur `default`. Farklı bir profile geçmek için, profil adını buradan belirtiniz `--profile` EFB'ye başlarken işaretleyin.

5.1 Yeni bir profil başlat

To create a new profile, you need to create a directory in the *`EFB_DATA_PATH`*`/profiles`, and create a new configuration file as described in chapter *Başlarken*.

Her şey düzgünce yapılandırıldığında, başlayabilirsiniz.

BÖLÜM 6

Support

6.1 Bug reports and feature requests

See *contribution guideline* for details.

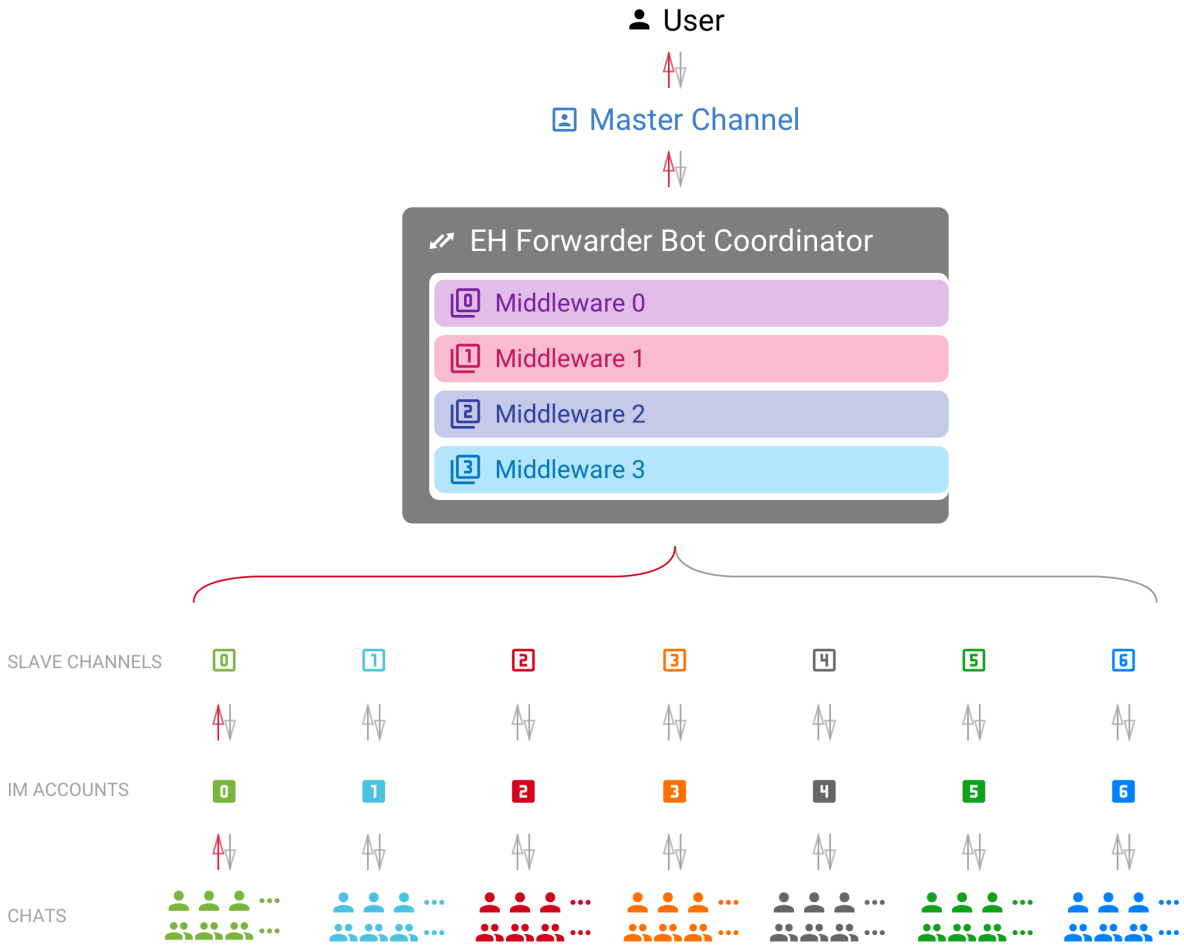
6.2 Questions about development and usage

If you have any question about developing a module for EFB, or about usages, you can always visit our GitHub Discussions forum or join our Telegram Group for help.

BÖLÜM 7

Walk-through — EFB nasıl çalışır?

EH Forwarder Bot, kullanıcıların farklı sohbet platformlarındaki hesaplarını birleştirilmiş bir arayüzde kontrol etmesine ve yönetmesine olanak tanıyan genişletilebilir bir çerçevedir. 4 bölümden oluşur: Ana Kanal, bazı Bağımlı Kanalları, bazı Ara Katman Yazılımları ve bir Koordinatör.



master channel The channel that directly interact with *the User*. It is guaranteed to have one and only one master channel in an EFB session.

slave channel The channel that delivers messages to and from their relative platform. There is at lease one slave channel in an EFB session.

coordinator Kanal örneklerini koruyan ve kanallar arasında mesajlar gönderen bir yapı bileşeni.

middleware Kanallar arasında iletilen mesajları ve durumları işleyen ve gerektiğinde değişiklikler yapan modül.

7.1 Bilinecek kavramlar

module A common term that refers to both channels and middlewares.

the User

the User Themself This term[1] can refer to the user of the current instance of EH Forwarder Bot, operating the master channel, and the account of an IM platform logged in by a slave channel.

chat A place where conversations happen, it can be either a *private chat*, a *group chat*, or a *system chat*.

[1] "Themself" here is used as a derived form of a gender-neutral singular third-person pronoun.

private chat A conversation with a single person on the IM platform. Messages from a private conversation shall only has an author of *the User Themself*, the other person, or a "system member".

For platforms that support bot or something similar, they would also be considered as a "user", unless messages in such chat can be sent from any user other than the bot.

For chats that *the User* receive messages, but cannot send message to, it should also be considered as a private chat, only to raise an exception when messages was trying to send to the chat.

group chat A chat that involves more than two members. A group chat MUST provide a list of members that is involved in the conversation.

system chat A chat that is a part of the system. Usually used for chats that are either a part of the IM platform, the *slave channel*, or a *middleware*. *Slave channel*s can use this chat type to send system message and notifications to the master channel.

chat member A participant of a chat. It can be *the User Themself*, another person or bot in the chat, or a virtual one created by the IM platform, the *slave channel*, or a *middleware*.

message Mesajlar tam olarak ana kanal ve bağımlı kanallar arasında iletilmektedir. Genellikle belirli bir türde bilgi taşırlar.

Her mesajın, en azından bağımlı kanala ait kendine özgü benzersiz bir kimliği olması gerekir. Düzenlenen herhangi bir mesaj aynı benzersiz kimlikle tanımlanabilmelidir.

status Bir mesaj biçimlendirilmemiş bilgi. Genellikle, sohbetlerin ve sohbet üyelerinin güncellenmesini ve mesajların kaldırılmasını içerir.

7.2 Bağımlı Kanallar

Bağımlı kanal işi nispeten basittir.

1. Ana kanala ve ana kanala mesaj gönderme.
2. Mevcut tüm sohbetlerin ve grup üyelerinin bir listesini korur.
3. Sohbetteki değişiklikleri izler ve ana kanala bildirir

Standart EFB Slave Kanal modeline uymayan özellikler *Ek özellikler*. olarak sunulabilir.

7.3 Ana Kanallar

Master channels is relatively more complicated and also more flexible. As it directly faces the User, its user interface should be user-friendly, or at least friendly to the targeted users.

Ana kanal işi şunları içerir:

1. Bağımlı kanallardan gelen mesajları alın, işleyin ve görüntüleyin.
2. Tüm bağımlı kanallardan gelen sohbetlerin tam listesini görüntüleyin.
3. Offer an interface for the User to use "extra functions" from slave channels.
4. Bağımlı kanallardaki güncellemeleri işle.
5. Mümkün olduğunda kullanıcı-dostu bir arayüz sağla.

7.4 Özel yazılımlar

Middlewares, kanallar arasında iletilen mesajları ve durumları izleyebilir, değiştirebilir veya geçersiz kılabilir. Middlewares, birbiri ardına, sırayla düzenlenir. Bir ara katman, varsa, önceki ara katman tarafından işlenen iletileri alacaktır. Bir katman, bir mesajı veya durumu geçersiz kıldığında, mesaj işlenmeyecektir ve daha fazla teslim edilmeyecektir.

BÖLÜM 8

Geliştirme rehberi

This section includes guides on how to develop channels and middlewares for EH Forwarder Bot.

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “NOT RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in BCP 14 [**RFC 2119**] [**RFC 8174**] when, and only when, they appear in all capitals, as shown here.

8.1 Bağımlı kanallar

Bağımlı kanal, IM’nin API’si üzerindeki bir pakete benziyor, IM’den gelen iletileri uygun nesnelere ekler ve onu ana kanala teslim eder.

Bağımlı kanalın bir IM platformuyla eşleşmesi gerektiğini önermemize rağmen, bilgileri mesaj olarak iletebilen her şey için modellemeyi deneyebilirsiniz, ve sohbetlere ve sohbetlerden mesaj iletmek için son noktaların sınırlı bir listesine sahiptir.

In most of the cases, slave channels SHOULD be identified as one single user from the IM platform, instead of a bot. You should only use a bot for slave channels when:

- IM platformu kullanıcı ve bot arasında fark etmez, veya
- iM platformundaki botlar, aynı şeyleri bir kullanıcı olarak yapabilir, eğer daha fazla değilse, bir kullanıcı olarak, botlar kullanıcı hesabından daha kolay oluşturulabilir.

8.1.1 Ek özellikler

Bağımlı kanallar, EFB'nin ihtiyaç duyduğu grup oluşturma, arkadaş arama vb. gibi *ek özellikler* yoluyla daha fazla işlev sunabilir.

Such features are accessed by the user in a CLI-like style. An "additional feature" method MUST only take one string parameter aside from `self`, and wrap it with *`extra()`* decorator. The `extra` decorator takes 2 arguments: `name` – a short name of the feature, and `desc` – a description of the feature with its usage.

`desc` SHOULD describe what the feature does and how to use it. It's more like the help text for an CLI program. Since method of invoking the feature depends on the implementation of the master channel, you SHOULD use "`{function_name}`" as its name in `desc`, and master channel will replace it with respective name depend on their implementation.

The method MUST in the end return a string, which will be shown to the user as its result, or `None` to notify the master channel there will be further interaction happen. Depending on the functionality of the feature, it may be just a simple success message, or a long chunk of results.

The callable MUST NOT raise any exception to its caller. Any exceptions occurred within should be `expected` and processed.

Callable name of such methods has a more strict standard than a normal Python 3 identifier name, for compatibility reason. An additional feature callable name MUST:

- büyük/küçük harfe duyarlı
- yalnız büyük ve küçük harfler, rakamlar ve altçizgi içerir.
- bir basamakla başlamaz.
- 1 ve 20 arasında bir uzunlukta olmalı
- *mümkün olduğunca kısa ve öz olun, fakat anlaşılabilir tutun*

It can be expressed in a regular expression as:

```
^[A-Za-z][A-Za-z0-9_]{0,19}$
```

An example is as follows:

```
@extra(name="Echo",
       desc="Return back the same string from input.\n"
            "Usage:\n"
            "    {function_name} text")
def echo(self, arguments: str = "") -> str:
    return arguments
```

8.1.2 Mesaj komutları

Mesaj komutları genellikle bağımlı kanallar tarafından gönderilir, böylece kullanıcılar gerekli eylemlere sahip mesajlara cevap verebilir.

Mesaj komutlarının yararlı olabileceği olası durumlar:

- Bir kartvizit alındığında arkadaş olarak ekleyin.
- Arkadaş talebi alındığında kabul veya reddedin.
- Vote to a voting message.

Bir mesaj, her biri aşağıdakileri içeren bir `liste` komutuyla eklenebilir:

- insan dostu bir isim,
- çağrılabilir bir isim,
- konumsal argümanlardan oluşan bir `list`` (*args``), ve
- anahtar kelime argümanlarından oluşan bir `dict`` (``**kwargs`)

When the User clicked the button, the corresponding method of your channel will be called with provided arguments.

Note that all such methods MUST return a `str` as a respond to the action from user, and they MUST NOT raise any exception to its caller. Any exceptions occurred within MUST be `expect`ed and processed.

8.1.3 Mesaj teslimi

Slave channels SHOULD deliver all messages that the IM provides, including what the User sent outside of this channel. But it SHOULD NOT deliver message sent from the master channel again back to the master channel as a new message.

8.1.4 Implementation details

See *SlaveChannel*.

8.2 Ana kanallar

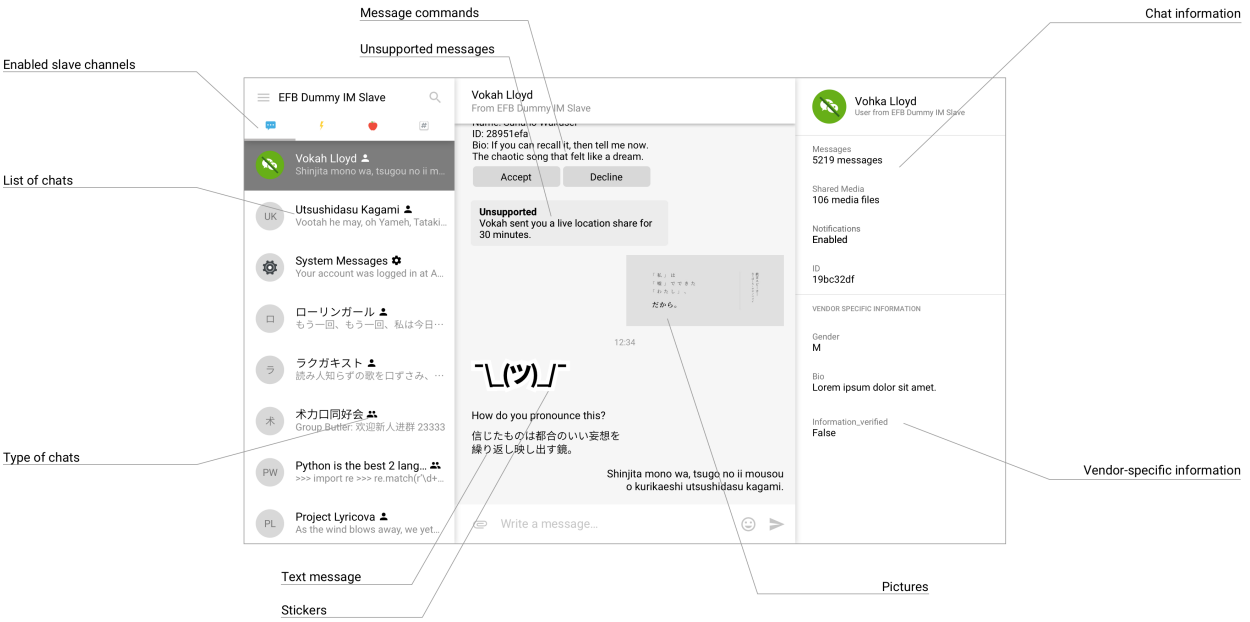
Ana kanallar, kullanıcı ile doğrudan veya dolaylı olarak etkileşim sağlayan arayüzdür. EFB'nin ilk ana kanalı (EFB Telegram Ana kanalı) Telegram Bot biçiminde yazılmasına rağmen, ana kanallar birçok biçimde yazılabilir, örneğin:

- Bir web uyulaması
- API'leri özel masaüstü bilgisayarlara ve mobil istemcilere gösteren bir sunucudur
- Mevcut IM'de sohbet botu
- Genel bir IM Protokolü ile derlenen sunucu
- Bir CLI istemcisi
- Aklınıza gelebilecek herşey…

8.2.1 Tasarım rehberi

When the master channel is implemented on an existing protocol or platform, as far as possible, while considering the user experience, a master channel SHOULD:

- maintain one conversation thread per chat, indicating its name, source channel and type;
- Sistemde tanımlanan çoğu türdeki iletileri desteklemek, kullanıcı ve slave kanalları arasında iletileri işlemek ve iletmek;
- support all, if not most, features of messages, including: targeted message reply, chat substitution in text (usually used in @ references), commands, etc. Master channel SHOULD be able to process incoming messages with such features, and send messages with such features to slave channels if applicable;
- bağımlı kanallar tarafından sunulan "ek özellikler" i çağırıp işleyebilir.



Şekil 1: Bir ana kanalın ideal tasarımının bir örneği, Telegram Masaüstünden esinlenilmiştir

İsteğe bağlı olarak, ana kanal, belirli bağımlı kanallardan satıcının belirttiği bilgileri destekleyebilir/tanımlayabilir.

Uygulamanıza bağlı olarak, bir ana kanalın sunum veya diğer amaçlar için büyük olasılıkla sohbet ve mesaj listesini koruması gerekebilir.

8.2.2 Mesaj teslimi

Note that sometimes the User may send messages outside of this EFB session, so that slave channels MAY provide a message with its author in the “self” type.

8.2.3 Implementation details

See `MasterChannel`.

8.3 Özel yazılımlar

Özel yazılımlar ana kanal ve bağımlı kanallar arasında çalışır, kanallar arasında iletilmiş mesajları ve durumları sıra sıra gözden geçirirler, devam etmelerini sağlarlar, değişiklikler yaparlar veya devre dışı bırakırlar.

Kanallarda olduğu gibi, orta sınıfların da EFB oturumu başına koordinatör tarafından yönetilen bir örneği olacaktır. Bununla birlikte, merkezi sorgulama mesaj dizisine sahip değillerdir, yani, bir ara katman yazılımı, bir sorgulama mesaj dizisi ya da arka planda benzer bir şey çalıştırmak istiyor ise, mesaj dizisi Python’un ``atexit``i kullanarak veya başka bir şekilde durdurmak zorunda kaldıkları anlamına gelir.

8.3.1 Mesaj ve Durum İşleme

Each middleware by default has 2 methods, *process_message()* which processes message objects, and *process_status()* which processes status objects. If they are not overridden, they will not touch on the object and pass it on as is.

To modify an object, just override the relative method and make changes to it. To discard an object, simply return `None`. When an object is discarded, it will not be passed further to other middlewares or channels, which means a middleware or a channel will never receive a `None` message or status.

8.3.2 Diğer Kullanımlar

Kanallara kıyasla birkaç kısıtlamaya sahip olup, mesaj ve durumların sadece kesişmesini sağlamaktan daha çok şey yapmaya izin veren özel yazılımları yazmak nispeten daha kolaydır.

Bazı fikirler:

- Ana / bağımlı kanallarına periyodik yayın
- Sohbet botları ile entegrasyon
- Satıcıya özel komutlarla ilgili otomatik işlemler / Ek özellikler
- Bağımlı kanallardan kullanıcı oturumlarını diğer programlarla paylaşma
- vb…

8.3.3 Implementation details

See *Middleware*.

8.4 Lifecycle

This section talks about the lifecycle of an EFB instance, and that of a message / status.

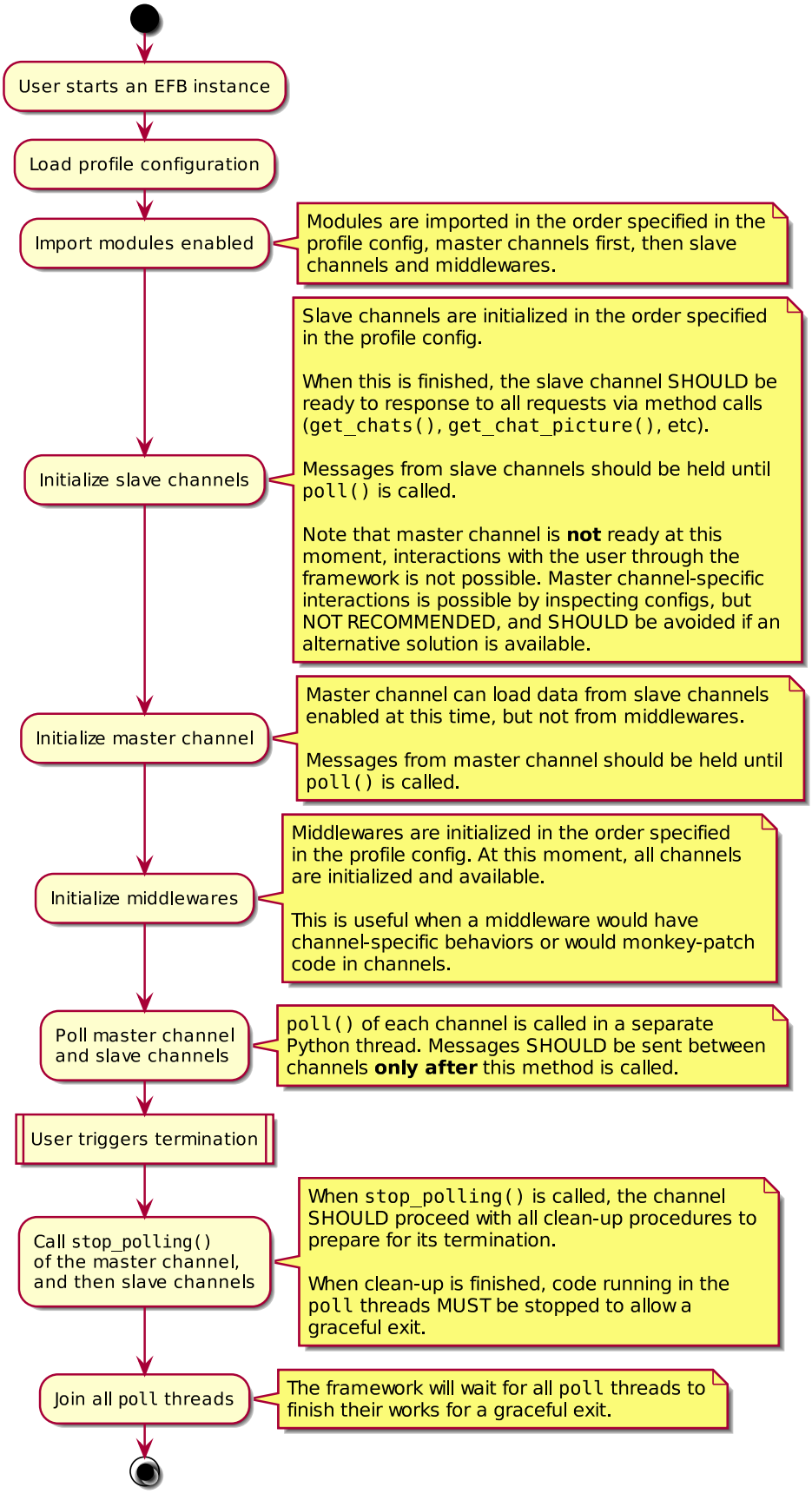
8.4.1 Lifecycle of an EFB instance

The diagram below outlines the lifecycle of an EFB instance, and how channels and middlewares are involved in it.

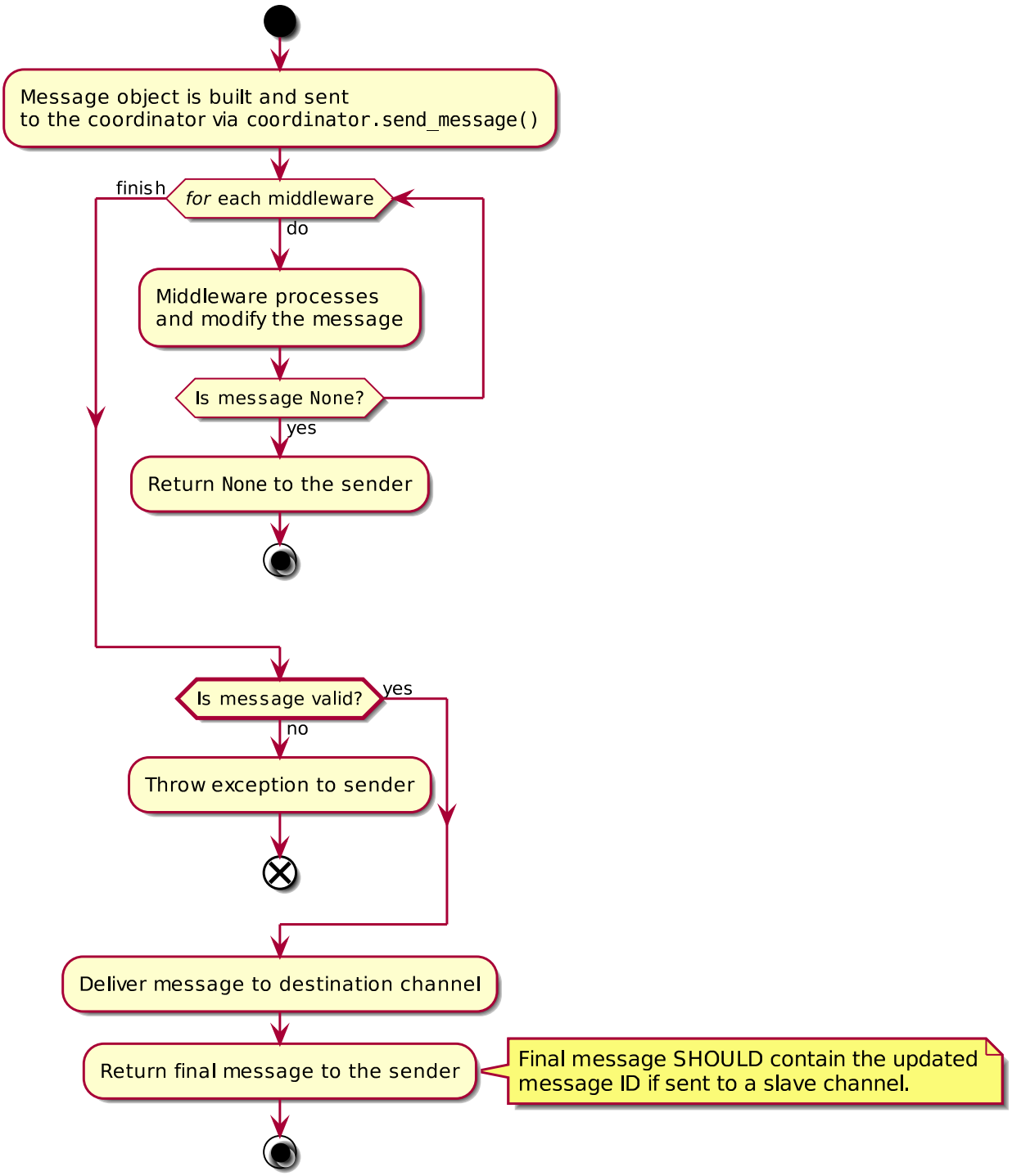
8.4.2 Lifecycle of a message

The diagram below outlines the lifecycle of a message sending from a channel, going through all middlewares, sent to the destination channel, and returned back to the sending channel.

Status objects processed in the same way.



Şekil 2: Lifecycle of an EFB instance



Şekil 3: Lifecycle of a message

8.5 Ortam işleme

8.5.1 Ortam formatı seçme

Hem Ana hem de Bağımlı Kanal aldıkları veya gönderdikleri medya dosyalarını dönüştürmede görev alabilirler. **Eğer uzaktan sunucudan alınan medya alışılmış formatta değilse, ulaştırmadan önce dönüştür; eğer uzaktan sunucuya gönderilen medyanın spesifik bir formatta olması gerekiyorsa; gönderilmeden önce dönüştürülmeli**. Yine de, bu, medya işleme hakkında kanalların sorumluluğunu ele alan tek kılavuzdur, ve herkesin ortak formatın / kodlamanın ne olduğu hakkında kendi fikri vardır. Bu nedenle biz bu davranımı sadece öneriyoruz, bizim kodlarımızı kullanmanız için sizi zorlamıyoruz. Başka bir deyişle, yine de gösterim metodunuzun yerini tuttuğu kabul edilmiş medya kodlaması tipini dikkate almalısınız ve eğer gerekiyorsa gösterim türünüzü farklı bir türe çevirmeli ve/veya geri çekmelisiniz. Sonuç olarak, bilginin aktarımı daha önemlidir.

8.5.2 Ortam kodlayıcılar

Benzer şekilde, bunun için de katı bir sınır koymayacağız, sadece bir öneri. Zaten bildiğiniz gibi, saf Python medya işleme kütüphaneleri azdır, bunların çoğu dahili veya harici ikili bağımlılık gerektirir.

Yapabildiğimiz kadar farklı birkaç belgeliği kullanmayı amaçlamayı deniyoruz, çünkü yüklenecek daha çok belgelik demek, daha çok alan, yükleme zamanı ve karışıklık demektir. Medya dosyalarını işlerken, eğer mümkünse aşağıdaki belgelikleri kullanmanızı öneriyoruz:

- Pillow
- FFmpeg

8.5.3 Mesajlardaki dosyalar

When a file sent out from a channel, it MUST be open, and sought back to 0 (`file.seek(0)`) before sending.

Files sent MUST be able to be located somewhere in the file system, and SHOULD with a appropriate extension name, but not required. All files MUST also have its MIME type specified in the message object. If the channel is not sure about the correct MIME type, it can try to guess with `libmagic`, or fallback to `application/octet-stream`. Always try the best to provide the most suitable MIME type when sending.

Bu gibi dosyalar için, biz ömrünün sonunu göstermek için "kapat" kelimesini kullandık. Eğer dosya artık gönderenin kanalı tarafından gerekli değilse, güvenli bir şekilde atılabilir.

Genel olarak, `tempfile.NamedTemporaryFile` olağan durumlar için çalışmalıdır.

8.6 Konfigürasyon ve depolama

8.6.1 Konfigürasyon ve Kalıcı Depolama

As described in *Altdizinler*, each module has been allocated with a folder per profile for configurations and other storage. The path can be obtained using `get_data_path()` with your module ID. All such storage is specific to only one profile.

Konfigürasyonlar için, bunu kullanmanızı öneririz `<module_data_path>/config.yaml`. Benzer olarak, bunu hazırladık :meth:`~.ehforwarderbot.utils.get_config_path`varsayılan konfig. dosyası için yolu alabilmek için. Yine, konfig. dosyanızın formatı için bu ismi veya YAML kullanmak için zorlanmıyorsunuz.

Genellikle depolama klasöründe bulunur:

- Yapılandırma dosyaları
- Kullanıcı kimlik bilgileri / Geçici depolama
- Veri tabanları

8.6.2 Geçici Depolama

Multimedya mesajları işlenirken, kaçınılmaz olarak, belirli dosyaları geçici olarak, kanal içinde veya kanallar arasında saklamalıyız. Genellikle, geçici dosyalar Python'un `tempfile` kütüphanesi ile işlenebilir.

8.6.3 Wizard

If your module requires relatively complicated configuration, it would be helpful to provide users with a wizard to *check prerequisites of your module* and *guide them to setup your module for use*.

From version 2, EFB introduced a centralised wizard program to allow users to enable or disable modules in a text-based user interface (TUI). If you want to include your wizard program as a part of the wizard, you can include a new entry point in your `setup.py` with Setuptools' Entry Point feature.

The group for wizard program is `ehforwarderbot.wizard`, and the entry point function MUST accept 2 positional arguments: profile ID and instance ID.

Example

`setup.py` script

```
setup(
    # ...
    entry_points={
        "ehforwarderbot.wizard": ['alice.irc = efb_irc_slave.wizard:main']
    },
    # ...
)
```

`.egg-info/entry_points.txt`

```
[ehforwarderbot.wizard]
alice.irc = efb_irc_slave.wizard:main
```

`efb_irc_slave/wizard.py`

```
# ...

def main(profile, instance):
    print("Welcome to the setup wizard of my channel.")
    print("You are setting up this channel in profile "
          "'{0}' and instance '{1}'.".format(profile, instance))
    print("Press ENTER/RETURN to continue.")
    input()

    step1()

    # ...
```

8.7 Derleme ve Yayınlama

8.7.1 Modülünüzü PyPI üzerinden yayınlayın

Kullanıcıların paketinizi yüklemelerinin en kolay yollarından biri, PyPI'de modülleri yayınlamaktır. Lütfen PyPI hakkında ilgili dokümanlara ve öğreticilere ve paketleri yayınlamak için pip'e bakınız.

For EFB modules, the package is RECOMMENDED to have a name starts with `efb-`, or in the format of `efb-platform-type`, e.g. `efb-irc-slave` or `efb-wechat-mp-filter-middleware`. If there is a collision of name, you MAY adjust the package name accordingly while keeping the package name starting with `efb-`.

When you are ready, you may also want to add your module to the Modules Repository of EFB.

8.7.2 Modül keşfi

EH Forwarder Bot, kanalları ve orta sayfaları keşfetmek ve yönetmek için *Setuptools' Entry Point feature* _ kullanır. `setup.py` komut dosyanıza veya `.egg-info/entry_points.txt` dizininde, grubu ve nesneyi aşağıdaki gibi belirtin:

- Ana kanallar için grup: `ehforwarderbot.master`
- Bağımlı kanallar için grup: `ehforwarderbot.slave`
- Middlewares için grup: `ehforwarderbot.middleware`

Convention for object names is `<author>.<platform>`, e.g. `alice.irc`. This MUST also be your module's ID.

Object reference MUST point to your module's class, which is a subclass of either *`Channel`* or *`Middleware`*.

8.7.3 Örnek

`setup.py` komut dosyası

```
setup(
    # ...
    entry_points={
        "ehforwarderbot.slave": ['alice.irc = efb_irc_slave:IRCChannel']
    },
    # ...
)
```

`.egg-info/entry_points.txt`

```
[ehforwarderbot.slave]
alice.irc = efb_irc_slave:IRCChannel
```

8.7.4 Özel modüller

Kendi kullanımınız için mevcut modüllerden uzatmak ya da değişiklikler yapmak isterseniz, modüllerinizi özel modüllere yerleştirebilirsiniz *directory*.

For such modules, your channel ID MUST be the fully-qualified name of the class. For example, if your class is located at `<EFB_BASE_PATH>/modules/bob_irc_mod/__init__.py:IRCChannel`, the channel MUST have ID `bob_ric_mod.IRCChannel` for the framework to recognise it.

8.8 Diğer

8.8.1 Günlük

In complex modules, you should have detailed logs in DEBUG level and optionally INFO level. All your log handlers SHOULD follow that of the root logger, which is controlled by the framework. This could be helpful when for you to locate issues reported by users.

8.8.2 Satıcı özellikleri

Mesajlara ve/veya sohbetlere satıcıya özgü bilgileri ekleyecekseniz, lütfen bunları README'inizde veya belgelerinizde belgelemek için gayret gösterin, böylece diğer geliştiriciler modülünüzü uyarlarken ona başvurabilirler.

8.8.3 İş parçacığı oluşturma

All channels are RECOMMENDED a separate thread while processing a new message, so as to prevent unexpectedly long thread blocking.

We are also considering to move completely to asynchronous programming when most channels are ready for the change.

8.8.4 Static type checking

EH Forwarder Bot is fully labeled in the Python 3 type hint notations. Since sometimes maintaining a module with high complexity could be difficult, we RECOMMEND you to type your module too and use tools like mypy to check your code statically.

BÖLÜM 9

Nasıl katkıda bulunulur

Öncelikle, katkıda bulunmak için zaman harcadığınızdan dolayı teşekkürler!

Please note that only questions on the framework will be answered here. For issue related with any channels, please contact their respective authors or post in their corresponding repositories.

Aşağıda, bir konuyla nasıl dosya indirebileceğinizle ilgili basit bir kılavuz veya yararlı ve etkili bir çekme isteği gönderin.

If you need help, or want to talk to the authors, feel free to visit our GitHub Discussions forum, or chat with us at our Telegram support group.

Before you ask a question, please read and follow this guide as far as possible. Without doing so might lead to unfriendly or no response from the community, although we try to refrain from doing so.

9.1 Arıza bildirme

9.1.1 Arıza raporunu bildirmeden önce

- Please ensure if your issue is about the framework itself, not about any module. Reports about modules should go to their respective issue trackers.
- Sorunuzu kapsamış olup olmadığını görmek için belgeleri okuyun.
- Bildirinizi görmek için current issue list 'i kontrol edin.

9.1.2 Nasıl (İyi) Bir Hata Raporu Gönderirim?

- Konuyla alakalı probleminizi tanımlarken **net ve betimleyici bir başlık kullanın**.
- Mümkün olduğunca detaylı olarak **probleme sebep olan adımları tarif ediniz**.
- **Adımları göstermek için spesifik örnekler temin ediniz.**
- **Adımları takip ettikten sonra gözlemlediğiniz davranışı tarif ediniz** ve bu davranışla ilgili problemin ne olduğunu tam olarak ifade ediniz.
- **Bunun yerine nasıl bir davranış görmeyi beklerdiniz ve neden.**
- **eğer problem spesifik bir işlemle tetiklenmemişse**, bu problem olmadan önce ne yaptığınızı betimleyiniz ve alttaki kılavuzu kullanarak daha çok bilgi paylaşın.
- ** Soruna ilişkin günlüğü sağlayın ** Günlüğe kaydetme işlemini başlatmak için ayrıntılı bayrağı kullanın ve yaptığınız ilk adımdan tüm günlük dosyasını gönderin.

Bu sorulara cevap vererek daha fazla içerik sağlayın:

- **Bu problem yakın zamanda mı oluşmaya başladı** (örneğin sürümü en sonuncuya güncelledikten sonra) ya da bu problem her zaman var mıydı?
- ** Sorunun güvenilir bir şekilde çoğaltılabilir misiniz? ** Değilse, sorunun ne sıklıkla yaşandığı ve hangi koşullar altında normalde gerçekleştiği hakkında ayrıntılı bilgi verin.

Konfigürasyonunuz ve ortamınız hakkında ayrıntı ekleyin:

- **What version of EFB are you using?** You can get the version by using the flag `--version`.
- **Kullandığınız OS'nin adı ve sürümü nedir?**

> **Dikkat:** Günlüğünüzü gönderirken lütfen kişisel bilgilerinizi gizlemeyi unutmayın.

9.2 Geliştirme önerilerinde bulunma

Eğer önerileriniz varsa, sorun listesinde belirtmekten çekinmeyin. Lütfen mümkün olduğunca çok detay vermeye çalışın, dahil edin:

- Öneriyi tanımlayan konu için **net ve betimleyici bir başlık kullanın**.
- ** Geliştirmenin nasıl davrandığına ilişkin ayrıntıları ver **.
- **Provide specific examples to demonstrate the abstraction**.
- The enhancement to the framework must be applicable to considerably many IM platforms, not just for a single IM. Suggestions for a specific IM should be made to their relative channel.

  GitHub Inc. tarafından Atom katkı rehberi 'dan uyarlanmıştır.

9.3 Çekme istekleri

Bazı değişiklikler yaptıysanız ve bize göndermek istiyorsanız, hesabınıza gönderin ve bir GitHub çekme isteği gönderin. Çekme isteğiniz için lütfen ayrıntılı bir açıklama yazın:

- **Ne değişiklikler yaptınız?**
- **Hangi problemleri çözdünüz?**
- ** Uygulanabilir ise ** hangi konuyu konuştunuz **.

İşlemleriniz için daima net bir günlük mesajı yazın. Tek satırlı mesajlar küçük değişiklikler için uygundur, ancak daha büyük değişikliklerin tek satırdan sonra ayrıntılı bir açıklamaya ihtiyacı vardır.

> Katılımcı Siyaset Vakfı tarafından hazırlanan OpenGovernment contribution guide dan uyarlanmıştır.

BÖLÜM 10

API documentations

Bu bölüm geçerli EH Forwarder Bot API'si için dokümantasyon içerir.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [**RFC 2119**] [**RFC 8174**] when, and only when, they appear in all capitals, as shown here.

10.1 Channel

class ehforwarderbot.channel.**Channel**(*instance_id =None*)

The abstract channel class.

channel_name

A human-friendly name of the channel.

Type str

channel_emoji

Emoji icon of the channel. Recommended to use a visually-length-one (i.e. a single grapheme cluster) emoji or other symbol that represents the channel best.

Type str

channel_id

Unique identifier of the channel. Convention of IDs is specified in *Derleme ve Yayınlama*. This ID will be appended with its instance ID when available.

Type *ModuleID* (str)

instance_id

The instance ID if available.

Type str

__init__(*instance_id =None*)

Initialize the channel. Inherited initializer MUST call the "super init" method at the beginning.

Parametreler **instance_id** (Optional[NewType()(InstanceID, str)]) – Instance ID of the channel.

get_message_by_id(*chat*, *msg_id*)

Get message entity by its ID. Applicable to both master channels and slave channels. Return None when message not found.

Override this method and raise *EFBOperationNotSupported* if it is not feasible to perform this for your platform.

Parametreler

- **chat** (*Chat*) – Chat in slave channel / middleware.
- **msg_id** (NewType()(MessageID, str)) – ID of message from the chat in slave channel / middleware.

Dönüş türü Optional[*Message*]

abstract poll()

Method to poll for messages. This method is called when the framework is initialized. This method SHOULD be blocking.

abstract send_message(*msg*)

Process a message that is sent to, or edited in this channel.

Notlar

Master channel MUST take care of the returned object that contains the updated message ID. Depends on the implementation of slave channels, the message ID MAY change even after being edited. The old message ID MAY be disregarded for the new one.

Parametreler **msg** (*Message*) – Message object to be processed.

Dönüşler The same message object. Message ID of the object MAY be changed by the slave channel once sent. This can happen even when the message sent is an edited message.

Dönüş türü *Message*

Harekete geçirir

- ***EFBChatNotFound*** – Raised when a chat required is not found.
- ***EFBMessageTypeNotSupported*** – Raised when the message type sent is not supported by the channel.
- ***EFBOperationNotSupported*** – Raised when an message edit request is sent, but not supported by the channel.
- ***EFBMessageNotFound*** – Raised when an existing message indicated is not found. E.g.: The message to be edited, the message referred in the msg.target attribute.
- ***EFBMessageError*** – Raised when other error occurred while sending or editing the message.

abstract send_status(*status*)

Process a status that is sent to this channel.

Parametreler **status** (*Status*) – the status object.

Harekete geçirir

- ***EFBChatNotFound*** – Raised when a chat required is not found.
- ***EFBMessageNotFound*** – Raised when an existing message indicated is not found. E.g.: The message to be removed.
- ***EFBOperationNotSupported*** – Raised when the channel does not support message removal.
- ***EFBMessageError*** – Raised when other error occurred while removing the message.

Not: Exceptions SHOULD NOT be raised from this method by master channels as it would be hard for a slave channel to process the exception.

This method is not applicable to Slave Channels.

---

stop_polling()
When EFB framework is asked to stop gracefully, this method is called to each channel object to stop all processes in the channel, save all status if necessary, and terminate polling.

When the channel is ready to stop, the polling function MUST stop blocking. EFB framework will quit completely when all polling threads end.

class ehforwarderbot.channel.**MasterChannel**(*instance_id =None*)
The abstract master channel class. All master channels MUST inherit this class.

class ehforwarderbot.channel.**SlaveChannel**(*instance_id =None*)
The abstract slave channel class. All slave channels MUST inherit this class.

supported_message_types
Types of messages that the slave channel accepts as incoming messages. Master channels may use this value to decide what type of messages to send to your slave channel.

Leaving this empty may cause the master channel to refuse sending anything to your slave channel.

Type Set[*MsgType*]

suggested_reactions
A list of suggested reactions to be applied to messages.

Reactions SHOULD be ordered in a meaningful way, e.g., the order used by the IM platform, or frequency of usage. Note that it is not necessary to list all suggested reactions if that is too long, or not feasible.

Set to `None` when it is known that no reaction is supported to ANY message in the channel. Set to empty list when it is not feasible to provide a list of suggested reactions, for example, the list of reactions is different for each chat or message.

Type Optional[Sequence[str]]

abstract get_chat(*chat_uid*)
Get the chat object from a slave channel.

Parametreler **chat_uid** (NewType()(ChatID, str)) – ID of the chat.

Dönüşler The chat found.

Dönüş türü .Chat

Harekete geçirir ***EFBChatNotFound*** – Raised when a chat required is not found.

abstract get_chat_picture(*chat*)
Get the profile picture of a chat. Profile picture is also referred as profile photo, avatar, "head image" sometimes.

Parametreler **chat** (*.Chat*) – Chat to get picture from.

Dönüşler

Opened temporary file object. The file object MUST have appropriate extension name that matches to the format of picture sent, and seek to position 0.

It MAY be deleted or discarded once closed, if not needed otherwise.

Dönüş türü BinaryIO

Harekete geçirir

- ***EFBChatNotFound*** – Raised when a chat required is not found.
- ***EFBOperationNotSupported*** – Raised when the chat does not offer a profile picture.

Örnekler

```
if chat.channel_uid != self.channel_uid:
    raise EFBChannelNotFound()
file = tempfile.NamedTemporaryFile(suffix=".png")
response = requests.post("https://api.example.com/get_profile_picture/png",
                         data={"uid": chat.uid})
if response.status_code == 404:
    raise EFBChatNotFound()
file.write(response.content)
file.seek(0)
return file
```

abstract get_chats()
Return a list of available chats in the channel.

Dönüşler a list of available chats in the channel.

Dönüş türü Collection[*Chat*]

get_extra_functions()
Get a list of additional features

Dönüş türü Dict[NewType()(ExtraCommandName, str), Callable]

Dönüşler A dict of methods marked as additional features. Method can be called with `get_extra_functions()["methodName"]()`.

10.1.1 Ortak işlemler

Mesajlar ve durumlar gönderiliyor

Sending messages and statuses to other channels is the most common operation of a channel. When enough information is gathered from external sources, the channel would then further process and pack them into the relevant objects, i.e. *Message* and *Status*.

When the object is built, the channel should sent it to the coordinator for following steps.

For now, both *Message* and *Status* has an attribute that indicates that where this object would be delivered to (`deliver_to` and *destination_channel*). This is used by the coordinator when delivering the message.

Messages MUST be sent using `coordinator.send_message()`. Statuses MUST be sent using `coordinator.send_status()`.

When the object is passed onto the coordinator, it will be further processed by the middleware and then to its destination.

For example, to send a message to the master channel

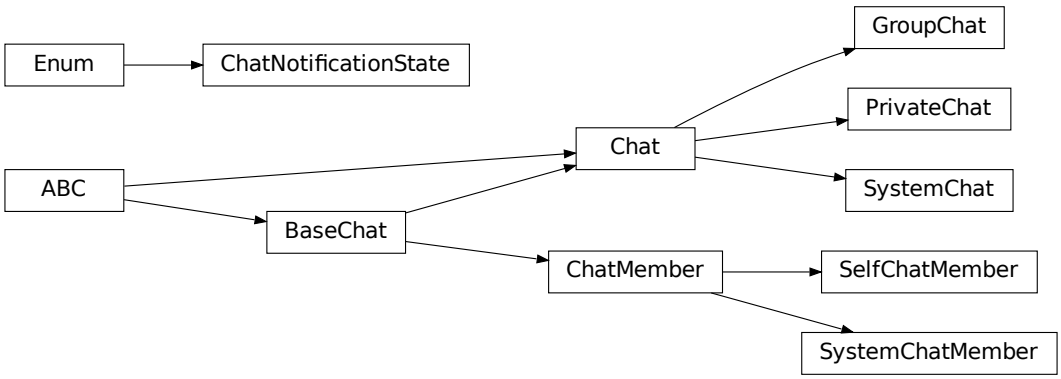
```
def on_message(self, data: Dict[str, Any]):
    """Callback when a message is received by the slave channel from
    the IM platform.
    """
    # Prepare message content ...
    message = coordinator.send_message(Message(
        chat=chat,
        author=author,
        type=message_type,
        text=text,
        # more details ...
        uid=data['uid'],
        deliver_to=coordinator.master
    ))
    # Post-processing ...
```

10.1.2 Kanal Kimliği Hakkında

Örnek kimliklerin sunulmasıyla birlikte, kanala atıfta bulunulurken sabit kodlanmış kimlik veya sabitler yerine `self.channel_id` veya eşdeğeri kullanmak gerekir (örn. Yapılandırma dosyalarına yol açarken sohbet oluştururken ve mesaj nesneleri vb.).

10.2 Chat and Chat Members

Inheritance diagram



Summary

PrivateChat(*[, channel, middleware, ...])	A private chat, where usually only the User Themself and the other participant are in the chat.
SystemChat(*[, channel, middleware, ...])	A system chat, where usually only the User Themself and the other participant (system chat member) are in the chat.
GroupChat(*[, channel, middleware, ...])	A group chat, where there are usually multiple members present.
ChatMember(chat, *[, name, alias, uid, id, ...])	Member of a chat.
SelfChatMember(chat, *[, name, alias, id, ...])	The User Themself as member of a chat.
SystemChatMember(chat, *[, name, alias, id, ...])	A system account/prompt as member of a chat.
ChatNotificationState(value)	Indicates the notifications settings of a chat in its slave channel or middleware.

Classes

class ehforwarderbot.chat.**BaseChat**(*, *channel =None*, *middleware =None*, *module_name =''*, *channel_emoji =''*, *module_id =''*, *name =''*, *alias =None*, *uid =''*, *id =''*, *vendor_specific =None*, *description =''*)

Bases: abc.ABC

Base chat class, this is an abstract class sharing properties among all chats and members. No instance can be created directly from this class.

Not: `BaseChat` objects are picklable, thus it is RECOMMENDED to keep any object of its subclass also picklable.

---

module_id
Unique ID of the module.

Type *ModuleID* (str)

channel_emoji
Emoji of the channel, empty string if the chat is from a middleware.

Type str

module_name
Name of the module.

Type *ModuleID* (str)

name
Name of the chat.

Type str

alias
Alternative name of the chat, usually set by user.

Type Optional[str]

uid
Unique ID of the chat. This MUST be unique within the channel.

Type *ChatID* (str)

description
A text description of the chat, usually known as “bio”, “description”, “purpose”, or “topic” of the chat.

Type str

vendor_specific
Any vendor specific attributes.

Type Dict[str, Any]

__init__(*, *channel =None*, *middleware =None*, *module_name =''*, *channel_emoji =''*, *module_id =''*, *name =''*, *alias =None*, *uid =''*, *id =''*, *vendor_specific =None*, *description =''*)

Parametreler

- **channel** (Optional[*SlaveChannel*]) – Provide the channel object to fill *module_name*, *channel_emoji*, and *module_id* automatically.
- **middleware** (Optional[*Middleware*]) – Provide the middleware object to fill *module_name*, and *module_id* automatically.
- **module_id** (NewType()(ModuleID, str)) – Unique ID of the module.
- **channel_emoji** (*str*) – Emoji of the channel, empty string if the chat is from a middleware.
- **module_name** (*str*) – Name of the module.
- **name** (*str*) – Name of the chat.
- **alias** (*Optional[str]*) – Alternative name of the chat, usually set by user.
- **uid** (NewType()(ChatID, str)) – Unique ID of the chat. This MUST be unique within the channel.
- **description** (*str*) – A text description of the chat, usually known as “bio”, “description”, “purpose”, or “topic” of the chat.
- **vendor_specific** (*Dict[str, Any]*) – Any vendor specific attributes.

copy()
Return a shallow copy of the object.

Dönüş türü TypeVar(_BaseChatSelf, bound = *BaseChat*, covariant =True)

property display_name: str
Shortcut property, equivalent to `alias or name`

Dönüş türü str

property long_name: str
Shortcut property, if alias exists, this will provide the alias with name in parenthesis. Otherwise, this will return the name

Dönüş türü str

abstract verify()
Verify the completeness of the data.

Harekete geçirir **AssertionError** – When this chat is invalid.

class ehforwarderbot.chat.**Chat**(*, *channel* =*None*, *middleware* =*None*, *module_name* ='', *channel_emoji* ='', *module_id* ='', *name* ='', *alias* =*None*, *id* ='', *uid* ='', *vendor_specific* =*None*, *description* ='', *members* =*None*, *notification* =*ChatNotificationState.ALL*, *with_self* =*True*)

Bases: *ehforwarderbot.chat.BaseChat*, abc.ABC

A chat object, indicates a user, a group, or a system chat. This class is abstract. No instance can be created directly from this class.

If your IM platform is providing an ID of the User Themself, and it is using this ID to indicate the author of a message, you SHOULD update *Chat.self.uid* accordingly.

```
>>> channel.my_chat_id
"david_divad"
>>> chat = Chat(channel=channel, name="Alice", uid=ChatID("alice123"))
>>> chat.self.uid = channel.my_chat_id
```

By doing so, you can get the author in one step:

```
author = chat.get_member(author_id)
```

… instead of using a condition check:

```
if author_id == channel.my_chat_id:
    author = chat.self
else:
    author = chat.get_member(author_id)
```

Not: Chat objects are picklable, thus it is RECOMMENDED to keep any object of its subclass also picklable.

---

module_id
Unique ID of the module.

Type *ModuleID* (str)

channel_emoji
Emoji of the channel, empty string if the chat is from a middleware.

Type str

module_name
Name of the module.

Type str

name
Name of the chat.

Type str

alias
Alternative name of the chat, usually set by user.

Type Optional[str]

uid
Unique ID of the chat. This MUST be unique within the channel.

Type *ChatID* (str)

description
A text description of the chat, usually known as “bio”, “description”, “purpose”, or “topic” of the chat.

Type str

notification
Indicate the notification settings of the chat in its slave channel (or middleware), defaulted to `ALL`.

Type *ChatNotificationState*

members
Provide a list of members in the chat. Defaulted to an empty `list`.

You can extend this object and implement a `@property` method set for loading members on demand.

Note that this list may include members created by middlewares when the object is a part of a message, and these members MAY not appear when trying to retrieve from the slave channel directly. These members would have a different *module_id* specified from the chat.

Type list of *ChatMember*

vendor_specific
Any vendor specific attributes.

Type Dict[str, Any]

self
the User as a member of the chat (if available).

Type Optional[*SelfChatMember*]

__init__(*, *channel =None*, *middleware =None*, *module_name =''*, *channel_emoji =''*, *module_id =''*, *name =''*, *alias =None*, *id =''*, *uid =''*, *vendor_specific =None*, *description =''*, *members =None*, *notification =ChatNotificationState.ALL*, *with_self =True*)

Keyword Arguments

- **module_id** (*str*) – Unique ID of the module.
- **channel_emoji** (*str*) – Emoji of the channel, empty string if the chat is from a middleware.
- **module_name** – Name of the module.
- **name** (*str*) – Name of the chat.
- **alias** (*Optional[str]*) – Alternative name of the chat, usually set by user.
- **id** – Unique ID of the chat. This MUST be unique within the channel.
- **description** (*str*) – A text description of the chat, usually known as “bio”, “description”, “purpose”, or “topic” of the chat.
- **notification** (ChatNotificationState) – Indicate the notification settings of the chat in its slave channel (or middleware), defaulted to `ALL`.
- **members** (MutableSequence[*ChatMember*]) – Provide a list of members of the chat. Defaulted to an empty `list`.
- **vendor_specific** (*Dict[str, Any]*) – Any vendor specific attributes.
- **with_self** (*bool*) – Initialize the chat with the User Themself as a member.

add_member(*name*, *uid*, *alias =None*, *id =''*, *vendor_specific =None*, *description =''*, *middleware =None*)
Add a member to the chat.

Tüyo: This method does not check for duplicates. Only add members with this method if you are sure that they are not added yet. To check if the member is already added before adding, you can do something like this:

```
with contextlib.suppress(KeyError):
    return chat.get_member(uid)
return chat.add_member(name, uid, alias=..., vendor_specific=...)
```

Parametreler

- **name** (*str*) – Name of the member.
- **uid** (NewType()(ChatID, str)) – ID of the member.

Keyword Arguments

- **alias** (*Optional[str]*) – Alias of the member.
- **vendor_specific** (*Dict[str, Any]*) – Any vendor specific attributes.
- **description** (*str*) – A text description of the chat, usually known as "bio", "description", "purpose", or "topic" of the chat.
- **middleware** (Optional[*Middleware*]) – Initialize this chat as a part of a middleware.

Dönüş türü *ChatMember*

add_self()
Add self to the list of members.

Harekete geçirir **AssertionError** – When there is already a self in the list of members.

Dönüş türü *SelfChatMember*

add_system_member(*name =''*, *alias =None*, *id =''*, *uid =''*, *vendor_specific =None*, *description =''*, *middleware =None*)
Add a system member to the chat.

Useful for slave channels and middlewares to create an author of a message from a system member when the "system" member is intended to become a member of the chat.

Tüyo: This method does not check for duplicates. Only add members with this method if you are sure that they are not added yet.

Keyword Arguments

- **name** (*str*) – Name of the member.
- **uid** – ID of the member.
- **alias** (*Optional[str]*) – Alias of the member.
- **vendor_specific** (*Dict[str, Any]*) – Any vendor specific attributes.
- **description** (*str*) – A text description of the chat, usually known as "bio", "description", "purpose", or "topic" of the chat.

- **middleware** (Optional[*Middleware*]) – Initialize this chat as a part of a middleware.

Dönüş türü *SystemChatMember*

get_member(*member_id*)
Find a member of chat by its ID.

Parametreler **member_id** (NewType()(ChatID, str)) – ID of the chat member.

Dönüş türü *ChatMember*

Dönüşler the chat member.

Harekete geçirir **KeyError** – when the ID provided is not found.

property has_self: bool
Indicate if this chat has yourself.

Dönüş türü bool

make_system_member(*name =''*, *alias =None*, *id =''*, *uid =''*, *vendor_specific =None*, *description =''*, *middleware =None*)
Make a system member for this chat.

Useful for slave channels and middlewares to create an author of a message from a system member when the "system" member is NOT intended to become a member of the chat.

Keyword Arguments

- **name** (*str*) – Name of the member.
- **uid** – ID of the member.
- **alias** (*Optional[str]*) – Alias of the member.
- **vendor_specific** (*Dict[str, Any]*) – Any vendor specific attributes.
- **description** (*str*) – A text description of the chat, usually known as "bio", "description", "purpose", or "topic" of the chat.
- **middleware** (Optional[*Middleware*]) – Initialize this chat as a part of a middleware.

Dönüş türü *SystemChatMember*

self: Optional[*ehforwarderbot.chat.SelfChatMember*]
The user as a member of the chat (if available).

class ehforwarderbot.chat.**ChatMember**(*chat*, *, *name =''*, *alias =None*, *uid =''*, *id =''*, *vendor_specific =None*, *description =''*, *middleware =None*)
Bases: *ehforwarderbot.chat.BaseChat*

Member of a chat. Usually indicates a member in a group, or the other participant in a private chat. Chat bots created by the users of the IM platform is also considered as a plain *ChatMember*.

To represent the User Themself, use *SelfChatMember*.

To represent a chat member that is a part of the system, the slave channel, or a middleware, use *SystemChatMember*.

*ChatMember*s MUST be created with reference of the chat it belongs to. Different objects MUST be created even when the same person appears in different groups or in a private chat.

*ChatMember*s are RECOMMENDED to be created using *Chat.add_member()* method.

Not: `ChatMember` objects are picklable, thus it is RECOMMENDED to keep any object of its subclass also picklable.

__init__(*chat*, ***, *name =''*, *alias =None*, *uid =''*, *id =''*, *vendor_specific =None*, *description =''*, *middleware =None*)

Parametreler **chat** (*Chat*) – Chat associated with this member.

Keyword Arguments

- **name** (*str*) – Name of the member.
- **alias** (*Optional[str]*) – Alternative name of the member, usually set by user.
- **uid** (*ChatID* (str)) – Unique ID of the member. This MUST be unique within the channel. This ID can be the same with a private chat of the same person.
- **description** (*str*) – A text description of the member, usually known as "bio", "description", "summary" or "introduction" of the member.
- **middleware** (*Middleware*) – Initialize this chat as a part of a middleware.

verify()

Verify the completeness of the data.

Harekete geçirir **AssertionError** – When this chat is invalid.

class ehforwarderbot.chat.**ChatNotificationState**(*value*)

Bases: enum.Enum

Indicates the notifications settings of a chat in its slave channel or middleware. If an exact match is not available, choose the most similar one.

ALL = -1

All messages in the chat triggers notifications.

MENTIONS = 1

Notifications are sent only when the User is mentioned in the message, in the form of @-references or quote-reply (message target).

NONE = 0

No notification is sent to slave IM channel at all.

class ehforwarderbot.chat.**GroupChat**(***, *channel =None*, *middleware =None*, *module_name =''*, *channel_emoji =''*, *module_id =''*, *name =''*, *alias =None*, *id =''*, *uid =''*, *vendor_specific =None*, *description =''*, *notification =ChatNotificationState.ALL*, *with_self =True*)

Bases: *ehforwarderbot.chat.Chat*

A group chat, where there are usually multiple members present.

Members can be added with the *add_member()* method.

If the `with_self` argument is `True` (which is the default setting), the User Themself would also be initialized as a member of the chat.

Örnekler

```
>>> group = GroupChat(channel=slave_channel, name="Wonderland", uid=ChatID(
 ↪"wonderland001"))
>>> group.add_member(name="Alice", uid=ChatID("alice"))
ChatMember(chat =<GroupChat: Wonderland (wonderland001) @ Example slave channel>,␣
 ↪name ='Alice', alias =None, uid ='alice', vendor_specific ={}, description ='')
>>> group.add_member(name="bob", alias="Bob James", uid=ChatID("bob"))
ChatMember(chat =<GroupChat: Wonderland (wonderland001) @ Example slave channel>,␣
 ↪name ='bob', alias ='Bob James', uid ='bob', vendor_specific ={}, description ='
 ↪')
>>> from pprint import pprint
>>> pprint(group.members)
[SelfChatMember(chat =<GroupChat: Wonderland (wonderland001) @ Example slave␣
 ↪channel>, name ='You', alias =None, uid ='__self__', vendor_specific ={},␣
 ↪description =''),
 ChatMember(chat =<GroupChat: Wonderland (wonderland001) @ Example slave channel>,
 ↪ name ='Alice', alias =None, uid ='alice', vendor_specific ={}, description ='
 ↪'),
 ChatMember(chat =<GroupChat: Wonderland (wonderland001) @ Example slave channel>,
 ↪ name ='bob', alias ='Bob James', uid ='bob', vendor_specific ={}, description =
 ↪'')]
```

Not: `GroupChat` objects are picklable, thus it is RECOMMENDED to keep any object of its subclass also picklable.

---

verify()

Verify the completeness of the data.

Harekete geçirir **AssertionError** – When this chat is invalid.

class ehforwarderbot.chat.**PrivateChat**(*, *channel =None*, *middleware =None*, *module_name =''*, *channel_emoji =''*, *module_id =''*, *name =''*, *alias =None*, *id =''*, *uid =''*, *vendor_specific =None*, *description =''*, *notification =ChatNotificationState.ALL*, *with_self =True*, *other_is_self =False*)

Bases: *ehforwarderbot.chat.Chat*

A private chat, where usually only the User Themself and the other participant are in the chat. Chat bots SHOULD also be categorized under this type.

There SHOULD only be at most one non-system member of the chat apart from the User Themself, otherwise it might lead to unintended behavior.

This object is by default initialized with the other participant as its member.

If the `with_self` argument is `True` (which is the default setting), the User Themself would also be initialized as a member of the chat.

Parametreler **other** – the other participant of the chat as a member

---

Not: `PrivateChat` objects are picklable, thus it is RECOMMENDED to keep any object of its subclass also picklable.

---

verify()

Verify the completeness of the data.

Harekete geçirir `AssertionError` – When this chat is invalid.

class ehforwarderbot.chat.**SelfChatMember**(*chat*, ***, *name =''*, *alias =None*, *id =''*, *uid =''*, *vendor_specific =None*, *description =''*, *middleware =None*)

Bases: *ehforwarderbot.chat.ChatMember*

The User Themself as member of a chat.

*SelfChatMember*s are RECOMMENDED to be created together with a chat object by setting `with_self` value to `True`. The created object is accessible at *Chat.self*.

The default ID of a *SelfChatMember* object is *SelfChatMember.SELF_ID*, and the default name is a translated version of the word "You".

You are RECOMMENDED to change the ID of this object if provided by your IM platform, and you MAY change the name or alias of this object depending on your needs.

Not: `SelfChatMember` objects are picklable, thus it is RECOMMENDED to keep any object of its subclass also picklable.

SELF_ID
The default ID of a *SelfChatMember*.

__init__(*chat*, ***, *name =''*, *alias =None*, *id =''*, *uid =''*, *vendor_specific =None*, *description =''*, *middleware =None*)

Parametreler **chat** (*Chat*) – Chat associated with this member.

Keyword Arguments

- **name** (*str*) – Name of the member.
- **alias** (*Optional[str]*) – Alternative name of the member, usually set by user.
- **uid** (*ChatID* (str)) – Unique ID of the member. This MUST be unique within the channel. This ID can be the same with a private chat of the same person.
- **description** (*str*) – A text description of the member, usually known as "bio", "description", "summary" or "introduction" of the member.
- **middleware** (*Middleware*) – Initialize this chat as a part of a middleware.

class ehforwarderbot.chat.**SystemChat**(***, *channel =None*, *middleware =None*, *module_name =''*, *channel_emoji =''*, *module_id =''*, *name =''*, *alias =None*, *id =''*, *uid =''*, *vendor_specific =None*, *description =''*, *notification =ChatNotificationState.ALL*, *with_self =True*)

Bases: *ehforwarderbot.chat.Chat*

A system chat, where usually only the User Themself and the other participant (system chat member) are in the chat. This object is used to represent system chat where the other participant is neither a user nor a chat bot of the remote IM.

Middlewares are RECOMMENDED to create chats with this type when they want to send messages in this type.

This object is by default initialized with the system participant as its member.

If the `with_self` argument is `True` (which is the default setting), the User Themself would also be initialized as a member of the chat.

Parametreler **other** – the other participant of the chat as a member

Not: `SystemChat` objects are picklable, thus it is RECOMMENDED to keep any object of its subclass also picklable.

verify()
Verify the completeness of the data.

Harekete geçirir **AssertionError** – When this chat is invalid.

class ehforwarderbot.chat.**SystemChatMember**(*chat*, ***, *name =''*, *alias =None*, *id =''*, *uid =''*, *vendor_specific =None*, *description =''*, *middleware =None*)

Bases: *ehforwarderbot.chat.ChatMember*

A system account/prompt as member of a chat.

Use this chat to send messages that is not from any specific member. Middlewares are RECOMMENDED to use this member type to communicate with the User in an existing chat.

Chat bots created by the users of the IM platform SHOULD NOT be created as a *SystemChatMember*, but a plain *ChatMember* instead.

*SystemChatMember*s are RECOMMENDED to be created using *Chat.add_system_member()* or *Chat.make_system_member()* method.

Not: `SystemChatMember` objects are picklable, thus it is RECOMMENDED to keep any object of its subclass also picklable.

SYSTEM_ID
The default ID of a *SystemChatMember*.

__init__(*chat*, ***, *name =''*, *alias =None*, *id =''*, *uid =''*, *vendor_specific =None*, *description =''*, *middleware =None*)

Parametreler **chat** (*Chat*) – Chat associated with this member.

Keyword Arguments

- **name** (*str*) – Name of the member.
- **alias** (*Optional[str]*) – Alternative name of the member, usually set by user.
- **uid** (*ChatID* (str)) – Unique ID of the member. This MUST be unique within the channel. This ID can be the same with a private chat of the same person.
- **description** (*str*) – A text description of the member, usually known as “bio”, “description”, “summary” or “introduction” of the member.
- **middleware** (*Middleware*) – Initialize this chat as a part of a middleware.

10.3 Sabitler

class ehforwarderbot.constants.**MsgType**(*value*)
Bir sayım.

Animation = 'Animation'
Message with an animation, usually in the form of GIF or soundless video.

Audio = 'Voice'
Audio messages (deprecated).

Use sürümünden beri kullanım dışı: *Voice* if the message has a voice message (usually recorded). Use *File* if the message has a music file (usually uploaded).

File = 'File'
File message.

Image = 'Image'
Image (picture) message.

Notlar

Animated GIF images must use *Animation* type instead.

Link = 'Link'
Başlıca tek bir bağlantı içeren mesaj, veya bir bağlantı önizlemesi içeren metin mesajı.

Location = 'Location'
Konum mesajı.

Status = 'Status'
Sohbetteki bir kullanıcının durumu, genellikle yazma ve yükleme.

Sticker = 'Sticker'
Birkaç metin yazısı ile gönderilmiş resim, genellikle şeffaf bir arkaplan, ve genellikle kullanıcının fotoğraf galerisinde bulunmayan sınırlı sayıda seçenekler.

Text = 'Text'
Metin mesajı

Unsupported = 'Unsupported'
Yukarıda listelenmeyen her tür mesaj. Metin gösterimi gereklidir.

Video = 'Video'
Video mesajı

Voice = 'Voice'
Voice messages, usually recorded right before sending.

10.4 Koordinatör

Coordinator among channels.

ehforwarderbot.coordinator.**profile**
Name of current profile..

Type str

ehforwarderbot.coordinator.**mutex**
Global interaction thread lock.

Type threading.Lock

ehforwarderbot.coordinator.**master**
The running master channel object.

Type *Channel*

ehforwarderbot.coordinator.**slaves**
Dictionary of running slave channel object. Keys are the unique identifier of the channel.

Type Dict[str, EFBChannel]

ehforwarderbot.coordinator.**middlewares**
List of middlewares

Type List[*Middleware*]

ehforwarderbot.coordinator.**add_channel**(*channel*)
Register the channel with the coordinator.

Parametreler **channel** (Channel) – Channel to register

ehforwarderbot.coordinator.**add_middleware**(*middleware*)
Register a middleware with the coordinator.

Parametreler **middleware** (Middleware) – Middleware to register

ehforwarderbot.coordinator.**get_module_by_id**(*module_id*)
Return the module instance of a provided module ID

Parametreler **module_id** (NewType()(ModuleID, str)) – Module ID, with instance ID if available.

Dönüş türü Union[*Channel*, *Middleware*]

Dönüşler Module instance requested.

Harekete geçirir **NameError** – When the module is not found.

ehforwarderbot.coordinator.**master: *ehforwarderbot.channel.MasterChannel***
The instance of the master channel.

ehforwarderbot.coordinator.**master_thread: Optional[threading.Thread] = None**
The thread running poll() of the master channel.

ehforwarderbot.coordinator.**middlewares:**
List[*ehforwarderbot.middleware.Middleware*] = []
Instances of middlewares. Sorted in the order of execution.

ehforwarderbot.coordinator.**mutex: _thread.allocate_lock = <unlocked**
_thread.lock object>
Mutual exclusive lock for user interaction through CLI interface

ehforwarderbot.coordinator.**profile: str = 'default'**
Current running profile name

ehforwarderbot.coordinator.**send_message**(*msg*)
Deliver a new message or edited message to the destination channel.

Parametreler **msg** (Message) – The message

Dönüş türü Optional[*Message*]

Dönüşler The message processed and delivered by the destination channel, includes the updated message ID if sent to a slave channel. Returns None if the message is not sent.

ehforwarderbot.coordinator.**send_status**(*status*)
Deliver a status to the destination channel.

Parametreler **status** (Status) – The status

ehforwarderbot.coordinator.**slave_threads: Dict[ModuleID, threading.Thread] = {}**
Threads running poll() from slave channels. Keys are the channel IDs.

ehforwarderbot.coordinator.**slaves: Dict[ModuleID, *ehforwarderbot.channel.SlaveChannel*] = {}**
Instances of slave channels. Keys are the channel IDs.

ehforwarderbot.coordinator.**translator: gettext.NullTranslations = <gettext.NullTranslations object>**
Internal GNU gettext translator.

10.5 Kural dışı durumlar

exception ehforwarderbot.exceptions.**EFBException**
Bases: Exception

A general class to indicate that the exception is from EFB framework.

exception ehforwarderbot.exceptions.**EFBChatNotFound**
Bases: *ehforwarderbot.exceptions.EFBException*

Raised by a slave channel when a chat indicated is not found.

Can be raised by any method that involves a chat or a message.

exception ehforwarderbot.exceptions.**EFBChannelNotFound**
Bases: *ehforwarderbot.exceptions.EFBException*

Raised by the coordinator when the message sent delivers to a missing channel.

exception ehforwarderbot.exceptions.**EFBMessageError**
Bases: *ehforwarderbot.exceptions.EFBException*

Raised by slave channel for any other error occurred when sending a message or a status.

Can be raised in *Channel.send_message()* and *Channel.send_status()*.

exception ehforwarderbot.exceptions.**EFBMessageNotFound**
Bases: *ehforwarderbot.exceptions.EFBMessageError*

Raised by a slave channel when a message indicated is not found.

Can be raised in *Channel.send_message()* (edited message / target message not found) and in *Channel.send_status()* (message to delete is not found).

exception ehforwarderbot.exceptions.**EFBMessageTypeNotSupported**
Bases: *ehforwarderbot.exceptions.EFBMessageError*

Raised by a slave channel when the indicated message type is not supported.

Can be raised in *Channel.send_message()*.

exception ehforwarderbot.exceptions.**EFBOperationNotSupported**
Bases: *ehforwarderbot.exceptions.EFBMessageError*

Raised by slave channels when a chat operation is not supported. E.g.: cannot edit message, cannot delete message.

Can be raised in *Channel.send_message()* and *Channel.send_status()*.

exception ehforwarderbot.exceptions.**EFBMessageReactionNotPossible**
Bases: *ehforwarderbot.exceptions.EFBException*

Raised by slave channel when a message reaction request from master channel is not possible to be processed.

Can be raised in *Channel.send_status()*.

10.6 Message

Summary

Message(*[, attributes, chat, author, ...])	A message.
LinkAttribute(title[, description, image, url])	Attributes for link messages.
LocationAttribute(latitude, longitude)	Attributes for location messages.
StatusAttribute(status_type[, timeout])	Attributes for status messages.
MessageCommands(commands)	Message commands.
MessageCommand(name, callable_name[, args, ...])	A message command.
Substitutions(substitutions)	Message text substitutions, or "@-references".

Classes

class ehforwarderbot.message.**Message**(*, *attributes =None*, *chat =None*, *author =None*, *commands =None*, *deliver_to =None*, *edit =False*, *edit_media =False*, *file =None*, *filename =None*, *is_system =False*, *mime =None*, *path =None*, *reactions =None*, *substitutions =None*, *target =None*, *text =''*, *type =MsgType.Unsupported*, *uid =None*, *vendor_specific =None*)

A message.

Not: `Message` objects are picklable, thus it is strongly RECOMMENDED to keep any object of its subclass also picklable.

Keyword Arguments

- **attributes** (Optional[*MessageAttribute*]) – Attributes used for a specific message type. Only specific message type requires this attribute. Defaulted to `None`.
  - Link: *LinkAttribute*

  - Location: *LocationAttribute*
  - Status: Typing/Sending files/etc.: *StatusAttribute*

Not: Do NOT use object of the abstract class *MessageAttribute* for `attributes`, but object of specific class instead.

- **chat** (*Chat*) – Sender of the message.
- **author** (*ChatMember*) – Author of this message. Author of the message MUST be indicated as a part of the same `chat` this message is from. If the message is sent from the User Themself, this MUST be an object of *SelfChatMember*.

  Note that the author MAY not be inside *members* of the chat of this message. The author MAY have a different *module_id* from the `chat`, and could be unretrievable otherwise.
- **commands** (Optional[*MessageCommands*]) – Commands attached to the message

  This attribute will be ignored in _Status_ messages.
- **deliver_to** (*Channel*) – The channel that the message is to be delivered to.
- **edit** (*bool*) – Flag this up if the message is edited. Flag only this if no multimedia file is modified, otherwise flag up both this one and `edit_media` as well.

  If no media file is modified, the edited message MAY carry no information about the file.

  This attribute will be ignored in _Status_ messages.
- **edit_media** (*bool*) – Flag this up if any file attached to the message is modified. If this value is true, `edit` MUST also be `True`. This attribute is ignored if the message type is not supposed to contain any media file, e.g. `Text`, `Location`, etc.

  This attribute will be ignored in _Status_ messages.
- **file** (*Optional[BinaryIO]*) – File object to multimedia file, type "rb". `None` if N/A. Recommended to use `NamedTemporaryFile`. The file SHOULD be able to be safely deleted (or otherwise discarded) once closed. All file object MUST be sought back to 0 (`file.seek(0)`) before sending.
- **filename** (*Optional[str]*) – File name of the multimedia file. `None` if N/A
- **is_system** (*bool*) – Mark as true if this message is a system message.
- **mime** (*Optional[str]*) – MIME type of the file. `None` if N/A
- **path** (*Optional[Path]*) – Local path of multimedia file. `None` if N/A
- **reactions** (Dict[str, Collection[`Chat`]]) – Indicate reactions to the message. Dictionary key is the canonical name of reaction, usually an emoji. Value is a collection of users who reacted to the message with that certain emoji. All `Chat` objects in this dict MUST be members in the chat of this message.

  This attribute will be ignored in _Status_ messages.
- **substitutions** (Optional[*Substitutions*]) – Substitutions of messages, usually used when the some parts of the text of the message refers to another user or chat.

  This attribute will be ignored in _Status_ messages.
- **target** (Optional[*Message*]) – Target message (usually for messages that "replies to" another message).

  This attribute will be ignored in _Status_ messages.

Not: This message MAY be a "minimum message", with only required fields:

- `Message.chat`
- `Message.author`
- `Message.text`
- `Message.type`
- `Message.uid`

- **text** (*str*) – Text of the message.

  This attribute will be ignored in _Status_ messages.

- **type** (*MsgType*) – Type of message
- **uid** (*str*) – Unique ID of message. Usually stores the message ID from slave channel. This ID MUST be unique among all chats in the same channel.

  Not: Some channels may not support message editing. Some channels may issue a new uid for edited message.

- **vendor_specific** (*Dict[str, Any]*) – A series of vendor specific attributes attached. This can be used by any other channels or middlewares that is compatible with such information. Note that no guarantee is provided for information in this section.

property link: Optional[*ehforwarderbot.message.LinkAttribute*]
Get the link attributes of the current message, if available.

Dönüş türü Optional[*LinkAttribute*]

property location: Optional[*ehforwarderbot.message.LocationAttribute*]
Get the location attributes of the current message, if available.

Dönüş türü Optional[*LocationAttribute*]

property status: Optional[*ehforwarderbot.message.StatusAttribute*]
Get the status attributes of the current message, if available.

Dönüş türü Optional[*StatusAttribute*]

verify()
Verify the validity of message.

Harekete geçirir **AssertionError** – when the message is not valid

class ehforwarderbot.message.**MessageAttribute**
Bases: abc.ABC

Abstract class of a message attribute.

class ehforwarderbot.message.**LinkAttribute**(*title*, *description =None*, *image =None*, *url =''*)
Bases: *ehforwarderbot.message.MessageAttribute*

Attributes for link messages.

title
Title of the link.

Type str

description
Description of the link.

Type str, optional

image
Image/thumbnail URL of the link.

Type str, optional

url
URL of the link.

Type str

__init__(*title*, *description =None*, *image =None*, *url =''*)

Parametreler

- **title** (*str*) – Title of the link.
- **description** (*str, optional*) – Description of the link.
- **image** (*str, optional*) – Image/thumbnail URL of the link.
- **url** (*str*) – URL of the link.

class ehforwarderbot.message.**LocationAttribute**(*latitude*, *longitude*)
Bases: *ehforwarderbot.message.MessageAttribute*

Attributes for location messages.

latitude
Latitude of the location.

Type float

longitude
Longitude of the location.

Type float

__init__(*latitude*, *longitude*)

Parametreler

- **latitude** (*float*) – Latitude of the location.
- **longitude** (*float*) – Longitude of the location.

class ehforwarderbot.message.**MessageCommand**(*name*, *callable_name*, *args =None*, *kwargs =None*)
Bases: object

A message command.

This object records a way to call a method in the module object. In case where the message has an `author` from a different module from the `chat`, this function MUST be called on the `author`'s module.

The method specified MUST return either a `str` as result or `None` if this message will be edited or deleted for further interactions.

name
Human-friendly name of the command.

Type str

callable_name
Callable name of the command.

Type str

args
Arguments passed to the function.

Type Collection[Any]

kwargs
Keyword arguments passed to the function.

Type Mapping[str, Any]

__init__(*name*, *callable_name*, *args =None*, *kwargs =None*)

Parametreler

- **name** (*str*) – Human-friendly name of the command.
- **callable_name** (*str*) – Callable name of the command.
- **args** (*Optional[Collection[Any]]*) – Arguments passed to the function. Defaulted to empty list;
- **kwargs** (*Optional[Mapping[str, Any]]*) – Keyword arguments passed to the function. Defaulted to empty dict.

class ehforwarderbot.message.**MessageCommands**(*commands*)
Bases: List[*ehforwarderbot.message.MessageCommand*]

Message commands.

Message commands allow user to take action to a specific message, including vote, add friends, etc.

commands
Commands for the message.

Type list of *MessageCommand*

__init__(*commands*)

Parametreler **commands** (list of *MessageCommand*) – Commands for the message.

class ehforwarderbot.message.**StatusAttribute**(*status_type*, *timeout =5000*)
Bases: *ehforwarderbot.message.MessageAttribute*

Attributes for status messages.

Message with type Status notifies the other end to update a chat-specific status, such as typing, send files, etc.

status_type
Type of status, possible values are defined in the StatusAttribute.

timeout
Number of milliseconds for this status to expire. Default to 5 seconds.

Type Optional[int]

Types
List of status types supported

class Types(*value*)
Bases: enum.Enum

TYPING
Used in *status_type*, represent the status of typing.

UPLOADING_FILE
Used in *status_type*, represent the status of uploading file.

UPLOADING_IMAGE
Used in *status_type*, represent the status of uploading image.

UPLOADING_VOICE
Used in *status_type*, represent the status of uploading voice.

UPLOADING_VIDEO
Used in *status_type*, represent the status of uploading video.

__init__ (*status_type*, *timeout =5000*)

Parametreler

- **status_type** (*Types*) – Type of status.
- **timeout** (*Optional[int]*) – Number of milliseconds for this status to expire. Default to 5 seconds.

class ehforwarderbot.message.**Substitutions**(*substitutions*)
Bases: Dict[Tuple[int, int], Union[*ehforwarderbot.chat.Chat*, *ehforwarderbot.chat.ChatMember*]]

Message text substitutions, or "@-references".

This is for the case when user "@-referred" a list of users in the message. Substitutions here is a dict of correspondence between the index of substring used to refer to a user/chat in the message and the chat object it referred to.

Values of the dictionary MUST be either a member of the chat (`self` or the other for private chats, group members for group chats) or another chat of the slave channel.

A key in this dictionary is a tuple of two `int`s, where first of it is the starting position in the string, and the second is the ending position defined similar to Python's substring. A tuple of `(3, 15)` corresponds to `msg.text[3:15]`. The value of the tuple `(a, b)` MUST satisfy $0 \leq a < b \leq l$, where l is the length of the message text.

Type: Dict[Tuple[int, int], *Chat*]

property is_mentioned: bool
Returns `True` if you are mentioned in this message.

In the case where a chat (private or group) is mentioned in this message instead of a group member, you will also be considered mentioned if you are a member of the chat.

Dönüş türü `bool`

10.6.1 Örnekler

Prelude: İlgili sohbetleri tanımlama

```
master: MasterChannel = coordinator.master
slave: SlaveChannel = coordinator.slave['demo.slave']
alice: PrivateChat = slave.get_chat("alice101")
bob: PrivateChat = slave.get_chat("bobrocks")
wonderland: GroupChat = slave.get_chat("thewonderlandgroup")
wonderland_alice: ChatMember = wonderland.get_member("alice101")
```

Başlatma ve işaretleme sohbetleri

1. A message delivered from slave channel to master channel

   ```
   message = Message(
       deliver_to=master,
       chat=wonderland,
       author=wonderland_alice,
       # More attributes go here...
   )
   ```

2. A message delivered from master channel to slave channel

   ```
   message = Message(
       deliver_to=slave,
       chat=alice,
       author=alice.self,
       # More attributes go here...
   )
   ```

Quoting a previous message (targeted message)

Data of the quoted message SHOULD be retrieved from recorded historical data. `Message.deliver_to` is not required for quoted message, and complete data is not required here. For details, see `Message.target`.

You MAY use the `Channel.get_message()` method to get the message object from the sending channel, but this might not always be possible depending on the implementation of the channel.

```
message.target = Message(
    chat=alice,
    author=alice.other,
    text="Hello, world.",
    type=MsgType.Text,
    uid=MessageID("100000002")
)
```

Önceden gönderilmiş mesajı düzenle

Message ID MUST be the ID from the slave channel regardless of where the message is delivered to.

```
message.edit = True
message.uid = MessageID("100000003")
```

Türe özgü bilgi

1. Metin mesajı

   ```
   message.type = MsgType.Text
   message.text = "Hello, Wonderland."
   ```

2. Medya mesajı

   Medya işleme ile ilgili bilgiler: doc:*/guide/media_processing* bölümünde açıklanmıştır.

   Aşağıdaki örnek resim(fotoğraf) mesajlar içindir. Ses, dosya, video, etiket aynı şekilde çalışır.

   In non-text messages, the `text` attribute MAY be an empty string.

   ```
   message.type = MsgType.Image
   message.text = "Image caption"
   message.file = NamedTemporaryFile(suffix=".png")
   message.file.write(binary_data)
   message.file.seek(0)
   message.filename = "holiday photo.png"
   message.mime = "image/png"
   ```

3. Konum mesajı

   In non-text messages, the `text` attribute MAY be an empty string.

   ```
   message.type = MsgType.Location
   message.text = "I'm here! Come and find me!"
   message.attributes = LocationAttribute(51.4826, -0.0077)
   ```

4. Bağlantı mesajı

   In non-text messages, the `text` attribute MAY be an empty string.

   ```
   message.type = MsgType.Link
   message.text = "Check it out!"
   message.attributes = LinkAttribute(
       title="Example Domain",
       description="This domain is established to be used for illustrative␣
    ↪examples in documents.",
       image="https://example.com/thumbnail.png",
       url="https://example.com"
   )
   ```

5. Durum

   In status messages, the `text` attribute is disregarded.

   ```
   message.type = MsgType.Status
   message.attributes = StatusAttribute(StatusAttribute.TYPING)
   ```

6. Desteklenmeyen mesaj

   Bu tür bir mesaj için `` text`` özelliği gereklidir.

```
message.type = MsgType.Unsupported
message.text = "Alice requested USD 10.00 from you. "
               "Please continue with your Bazinga App."
```

Ek bilgi

1. Değişim

   @-reference the User Themself, another member in the same chat, and the entire chat in the message text.

```
message.text = "Hey @david, @bob, and @all. Attention!"
message.substitutions = Substitutions({
    # text[4:10] == "@david", here David is the user.
    (4, 10): wonderland.self,
    # text[12:16] == "@bob", Bob is another member of the chat.
    (12, 16): wonderland.get_member("bob"),
    # text[22:26] == "@all", this calls the entire group chat, hence the
    # chat object is set as the following value instead.
    (22, 26): wonderland
})
```

2. Komutlar

```
message.text = "Carol sent you a friend request."
message.commands = MessageCommands([
    EFBCommand(name="Accept", callable_name="accept_friend_request",
               kwargs={"username": "carol_jhonos", "hash": "2a9329bd93f"}
 ↪),
    EFBCommand(name="Decline", callable_name="decline_friend_request",
               kwargs={"username": "carol_jhonos", "hash": "2a9329bd93f"})
])
```

10.7 Middleware

class ehforwarderbot.**Middleware**(*instance_id =None*)

Middleware class.

middleware_id

Unique ID of the middleware. Convention of IDs is specified in *Derleme ve Yayınlama*. This ID will be appended with its instance ID when available.

Type str

middleware_name

Human-readable name of the middleware.

Type str

instance_id

The instance ID if available.

Type str

__init__(*instance_id =None*)
Initialize the middleware. Inherited initializer MUST call the "super init" method at the beginning.

Parametreler **instance_id** (Optional[NewType()(InstanceID, str)]) – Instance ID of the middleware.

get_extra_functions()
Get a list of additional features

Dönüşler A dict of methods marked as additional features. Method can be called with `get_extra_functions()["methodName"]()`.

Dönüş türü Dict[str, Callable]

process_message(*message*)
Process a message with middleware

Parametreler **message** (*Message*) – Message object to process

Dönüşler Processed message or None if discarded.

Dönüş türü Optional[*Message*]

process_status(*status*)
Process a status update with middleware

Parametreler **status** (*Status*) – Message object to process

Dönüşler Processed status or None if discarded.

Dönüş türü Optional[*Status*]

10.7.1 Middleware Kimliği Hakkında

With the introduction of instance IDs, it is required to use the `self.middleware_id` or equivalent instead of any hard-coded ID or constants while referring to the middleware ID (e.g. while retrieving the path to the configuration files, etc).

10.7.2 Accept commands from user through Master Channel

Despite we do not limit how the User interact with your middleware, there are 2 common ways to do it through a master channel.

Capture messages

If the action is chat-specific, you can capture messages with a specific pattern. Try to make the pattern easy to type but unique enough so that you don't accidentally catch messages that were meant to sent to the chat.

You may also construct a virtual chat or chat member of type "System" to give responses to the User.

"Additional features"

If the action is not specific to any chat, but to the system as a whole, we have provided the same command line-like interface as in slave channels to middlewares as well. Details are available at *Ek özellikler*.

10.7.3 Chat-specific interactions

Middlewares can have chat-specific interactions through capturing messages and reply to them with a chat member created by the middleware.

The following code is an example of a middleware that interact with the user by capturing messages.

When the master channel sends a message with a text starts with `time`, the middleware captures this message and reply with the name of the chat and current time on the server. The message captured is not delivered to any following middlewares or the slave channel.

```
def process_message(self: Middleware, message: Message) -> Optional[Message]:
    if message.deliver_to != coordinator.master and \  # sent from master channel
        text.startswith('time`'):

        # Make a system chat object.
        # For difference between `make_system_member()` and `add_system_member()`,
        # see their descriptions above.
        author = message.chat.make_system_member(
            uid="__middleware_example_time_reporter__",
            name="Time reporter",
            middleware=self
        )

        # Make a reply message
        reply = Message(
            uid=f"__middleware_example_{uuid.uuid4()}__",
            text=f"Greetings from chat {message.chat.name} on {datetime.now().
↪strftime('%c')}.",
            chat=chat,
            author=author,  # Using the new chat we created before
            type=MsgType.Text,
            target=message,  # Quoting the incoming message
            deliver_to=coordinator.master  # message is to be delivered to master
        )
        # Send the message back to master channel
        coordinator.send_message(reply)

        # Capture the message to prevent it from being delivered to following␣
↪middlewares
        # and the slave channel.
        return None

    # Continue to deliver messages not matching the pattern above.
    return message
```

10.8 Durum

class ehforwarderbot.status.**Status**
Abstract class of a status

destination_channel
The channel that this status is sent to, usually the master channel.

Type *Channel*

class ehforwarderbot.status.**ChatUpdates**(*channel*, *new_chats =()*, *removed_chats =()*, *modified_chats =()*)
Inform the master channel on updates of slave chats.

channel
Slave channel that issues the update

Type *SlaveChannel*

new_chats
Unique ID of new chats

Type Optional[Collection[str]]

removed_chats
Unique ID of removed chats

Type Optional[Collection[str]]

modified_chats
Unique ID of modified chats

Type Optional[Collection[str]]

__init__(*channel*, *new_chats =()*, *removed_chats =()*, *modified_chats =()*)

Parametreler

- **channel** (*SlaveChannel*) – Slave channel that issues the update
- **new_chats** (*Optional[Collection[str]]*) – Unique ID of new chats
- **removed_chats** (*Optional[Collection[str]]*) – Unique ID of removed chats
- **modified_chats** (*Optional[Collection[str]]*) – Unique ID of modified chats

class ehforwarderbot.status.**MemberUpdates**(*channel*, *chat_id*, *new_members =()*, *removed_members =()*, *modified_members =()*)
Inform the master channel on updates of members in a slave chat.

channel
Slave channel that issues the update

Type *SlaveChannel*

chat_id
Unique ID of the chat.

Type str

new_members
Unique ID of new members

Type Optional[Collection[str]]

removed_members
Unique ID of removed members

Type Optional[Collection[str]]

modified_members
Unique ID of modified members

Type Optional[Collection[str]]

__init__(*channel*, *chat_id*, *new_members =()*, *removed_members =()*, *modified_members =()*)

Parametreler

- **channel** (*SlaveChannel*) – Slave channel that issues the update
- **chat_id** (*str*) – Unique ID of the chat.
- **new_members** (*Optional[Collection[str]]*) – Unique ID of new members
- **removed_members** (*Optional[Collection[str]]*) – Unique ID of removed members
- **modified_members** (*Optional[Collection[str]]*) – Unique ID of modified members

class ehforwarderbot.status.**MessageReactionsUpdate**(*chat*, *msg_id*, *reactions*)
Update reacts of a message, issued from slave channel to master channel.

Parametreler

- **chat** (*Chat*) – The chat where message is sent
- **msg_id** (*str*) – ID of the message for the reacts
- **reactions** (Mapping[NewType()(ReactionName, str), Collection[*ChatMember*]]) – Indicate reactions to the message. Dictionary key represents the reaction name, usually an emoji. Value is a collection of users who reacted to the message with that certain emoji. All Chat objects in this dict MUST be members in the chat of the message.
- **destination_channel** (*MasterChannel*) – Channel the status is issued to, which is always the master channel.

__init__(*chat*, *msg_id*, *reactions*)

Parametreler

- **chat** (*Chat*) – The chat where message is sent
- **msg_id** (*str*) – ID of the message for the reacts
- **reactions** (Mapping[NewType()(ReactionName, str), Collection[*ChatMember*]]) – Indicate reactions to the message. Dictionary key represents the reaction name, usually an emoji. Value is a collection of users who reacted to the message with that certain emoji. All Chat objects in this dict MUST be members in the chat of the message.

class ehforwarderbot.status.**MessageRemoval**(*source_channel*, *destination_channel*, *message*)
Inform a channel to remove a certain message.

This is usually known as “delete from everyone”, “delete from recipient”, “recall a message”, “unsend”, or “revoke a message” as well, depends on the IM platform.

Some channels MAY not support removal of messages, and raises a *exceptions.EFBOperationNotSupported* exception.

Feedback by sending another `MessageRemoval` back is not required when this object is sent from a master channel. Master channels SHOULD treat a successful delivery of this status as a successful removal.

source_channel
Channel issued the status

Type *Channel*

destination_channel
Channel the status is issued to

Type *Channel*

message
Message to remove. This MAY not be a complete *message.Message* object.

Type *Message*

Harekete geçirir **.exceptions.EFBOperationNotSupported** – When message removal is not supported in the channel.

__init__(*source_channel*, *destination_channel*, *message*)
Create a message removal status

Try to provided as much as you can, if not, provide a minimum information in the channel:

- Slave channel ID and chat ID (*message.chat.module_id* and *message.chat.uid*)
- Message unique ID from the slave channel (`message.uid`)

Parametreler

- **source_channel** (*Channel*) – Channel issued the status
- **destination_channel** (*Channel*) – Channel the status is issued to
- **message** (*Message*) – Message to remove.

class ehforwarderbot.status.**ReactToMessage**(*chat*, *msg_id*, *reaction*)
Created when user react to a message, issued from master channel.

When this status is sent, a *status.MessageReactionsUpdate* is RECOMMENDED to be issued back to master channel.

Parametreler

- **chat** (`Chat`) – The chat where message is sent
- **msg_id** (*str*) – ID of the message to react to
- **reaction** (*Optional[str]*) – The reaction name to be sent, usually an emoji. Set to `None` to remove reaction.
- **destination_channel** (*SlaveChannel*) – Channel the status is issued to, extracted from the chat object.

Harekete geçirir

- **.exceptions.EFBMessageReactionNotPossible** – Raised when the reaction is not valid (e.g. the specific reaction is not in the list of possible reactions).
- **.exceptions.EFBOperationNotSupported** – Raised when reaction in any form is not supported to the message at all.

__init__(*chat*, *msg_id*, *reaction*)

Parametreler

- **chat** (*Chat*) – The chat where message is sent
- **msg_id** (*str*) – ID of the message to react to
- **reaction** (Optional[NewType() (ReactionName, str)]) – The reaction name to be sent, usually an emoji

10.9 Custom Type Hints

A list of type aliases when no separate class is defined for some types of values. Types for user-facing values (display names, descriptions, message text, etc.) are not otherwise defined.

Most of types listed here are defined under the "NewType" syntax in order to clarify some ambiguous values not covered by simple type checking. This is only useful if you are using static type checking in your development. If you are not using type checking of any kind, you can simply ignore values in this module.

ehforwarderbot.types.**ChatID**
Chat ID from slave channel or middleware, applicable to both chat and chat members.

alias of str

ehforwarderbot.types.**ExtraCommandName**
Command name of additional features, in the format of ^[A-Za-z][A-Za-z0-9_]{0,19}$.

alias of str

ehforwarderbot.types.**InstanceID**
Instance ID of a module.

alias of str

ehforwarderbot.types.**MessageID**
Message ID from slave channel or middleware.

alias of str

ehforwarderbot.types.**ModuleID**
Module ID, including instance ID after # if available.

alias of str

ehforwarderbot.types.**ReactionName**
Canonical representation of a reaction, usually an emoji.

alias of str

ehforwarderbot.types.**Reactions**
Reactions to a message.

alias of Mapping[*ReactionName*, Collection[ChatMember]]

10.10 Araçlar

ehforwarderbot.utils.**extra**(*name*, *desc*)
Decorator for slave channel's "additional features" interface.

Parametreler

- **name** (str) – A human readable name for the function.
- **desc** (str) – A short description and usage of it. Use {function_name} in place of the function name in the description.

Dönüş türü Callable[..., Optional[str]]

Dönüşler The decorated method.

Örnek

```
@extra(name="Echo", desc="Return the text entered.\n\nUsage:\n    {function_name} text")
def echo(self, text: str) -> Optional[str]:
    return text
```

ehforwarderbot.utils.**get_base_path**()
Get the base data path for EFB. This can be defined by the environment variable EFB_DATA_PATH.

If EFB_DATA_PATH is not defined, this gives ~/.ehforwarderbot.

This method creates the queried path if not existing.

Dönüş türü Path

Dönüşler The base path.

ehforwarderbot.utils.**get_config_path**(*module_id=None*, *ext='yaml'*)
Get path for configuration file. Defaulted to ~/.ehforwarderbot/profiles/*profile_name*/*module_id*/config.yaml.

This method creates the queried path if not existing. The config file will not be created, however.

Parametreler

- **module_id** (Optional[NewType()(ModuleID, str)]) – Module ID.
- **ext** (str) – Extension name of the config file. Defaulted to "yaml".

Dönüş türü Path

Dönüşler The path to the configuration file.

ehforwarderbot.utils.**get_custom_modules_path**()
Get the path to custom channels

Dönüş türü Path

Dönüşler The path for custom channels.

ehforwarderbot.utils.**get_data_path**(*module_id*)
Get the path for permanent storage of a module.

This method creates the queried path if not existing.

Parametreler **module_id** (NewType()(ModuleID, str)) – Module ID

Dönüş türü Path

Dönüşler The data path of indicated module.

ehforwarderbot.utils.**locate_module**(*module_id*, *module_type =None*)
Locate module by module ID

Parametreler

- **module_id** (NewType()(ModuleID, str)) – Module ID
- **module_type** (Optional[str]) – Type of module, one of 'master', 'slave' and 'middleware'

BÖLÜM 11

Dizinler ve tablolar

- genindex
- modindex
- search

BÖLÜM 12

Katkıda bulunmak istiyor musunuz?

Herkes bir konuyu gündeme getirmek ya da bir çekme isteği göndermekten memnuniyet duyar, bunu yapmadan önce *katılım kılavuzunu* boyunca okumayı ve unutmamayı unutmayın.

BÖLÜM 13

İlgili makaleler

- Idea: Group Chat Tunneling (Sync) with EH Forwarder Bot
- What's so new in EH Forwarder Bot 2 (and its modules)

İp uçları, tüyolar ve topluluğun katkı sağladığı makaleler için, bakınız project wiki.

BÖLÜM 14

Lisans

EFB framework is licensed under GNU Affero General Public License 3.0 or later versions.

```
EH Forwarder Bot: An extensible message tunneling chat bot framework.
Copyright (C) 2016 - 2020 Eana Hufwe, and the EH Forwarder Bot contributors
All rights reserved.

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 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 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 <http://www.gnu.org/licenses/>.
```

Bağış.

Python Modül Dizini

e

Dizin

Semboller

A

B

C

D

E

F

G

H

I

K

L

M

N

P

R

S

T

U

V