aiogram-i18n

Release 1.4

RootShinobi

Feb 27, 2024

CONTENTS:

aiogram_i18n, an unrivalled middleware for Telegram bot internationalization, breathes life into bots, enabling diverse interactions in multiple languages, while adeptly managing user context, paving the way for a truly engaging and immersive user experience irrespective of language preference, and providing robust support for both Fluent and GNU gettext localization systems, thereby offering flexibility in translation file format selection, streamlining the translation process, and making the creation of multilingual bots an achievable goal for developers.

CHAPTER

ONE

INSTALLATION

To install aiogram-i18n without any backends:

```
pip install aiogram-i18n
```

If you need help with what backend to choose, read *Backends*.

1.1 To use FluentCompileCore

To use fluent-compiler backend (*aiogram_i18n.cores.fluent_compile_core.FluentCompileCore*) install it:

```
pip install aiogram-i18n[compiler]
```

1.2 To use FluentRuntimeCore

To use Fluent.runtime backend (*aiogram_i18n.cores.fluent_runtime_core.FluentRuntimeCore*) install it:

```
pip install aiogram-i18n[runtime]
```

1.3 To use GNU gettext

To use gettext backend (`aiogram_i18n.cores.gnu_text_core.py.GettextCore`) install it:

```
pip install aiogram-i18n[gettext]
```

CHAPTER

TWO

USAGE EXAMPLE

```python
import asyncio
from contextlib import suppress
from logging import INFO, basicConfig
from typing import Any

from aiogram import Bot, Dispatcher, Router
from aiogram.enums import ParseMode
from aiogram.filters import CommandStart
from aiogram.types import Message

from aiogram_i18n import I18nContext, I18nMiddleware, LazyProxy
from aiogram_i18n.cores.fluent_runtime_core import FluentRuntimeCore
from aiogram_i18n.lazy.filter import LazyFilter
from aiogram_i18n.types import (
    KeyboardButton,
    ReplyKeyboardMarkup,
)  # you should import mutable objects from here if you want to use LazyProxy in them

router = Router(name=__name__)
rkb = ReplyKeyboardMarkup(
    keyboard=[[KeyboardButton(text=LazyProxy("help"))]], resize_keyboard=True  # or L.help()
)


@router.message(CommandStart())
async def cmd_start(message: Message, i18n: I18nContext) -> Any:
    name = message.from_user.mention_html()
    return message.reply(
        text=i18n.get("hello", user=name), reply_markup=rkb  # or i18n.hello(user=name)
    )


@router.message(LazyFilter("help"))
async def cmd_help(message: Message) -> Any:
    return message.reply(text="-- " + message.text + " --")


async def main() -> None:
    basicConfig(level=INFO)
```

(continues on next page)

(continued from previous page)

```
    bot = Bot("42:ABC", parse_mode=ParseMode.HTML)
    i18n_middleware = I18nMiddleware(core=FluentRuntimeCore(path="locales/{locale}/LC_
↪MESSAGES"))

    dp = Dispatcher()
    dp.include_router(router)
    i18n_middleware.setup(dispatcher=dp)

    await dp.start_polling(bot)


if __name__ == "__main__":
    with suppress(KeyboardInterrupt):
        asyncio.run(main())
```

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

3.1 Backends

aiogram-i18n supports many backends of translation and localization systems, so you can use any of them to localize your bot.

3.1.1 Fluent runtime

Fluent.runtime is a part of Project Fluent, which is a localization system developed by Mozilla for natural-sounding translations. It's designed to unleash the expressive power of the natural language.

If you want to use Fluent it's recommended to use *fluent_compile* instead of runtime as it's the most efficient way to use Fluent in Python.

Pros of using fluent.runtime

- Fluent leverages Unicode and handles complex grammatical cases and gender.
- It works on a wide range of platforms and operating systems.
- The fluent syntax is very readable which makes it easier for localizers to understand.
- It handles placeholders and markup with ease.

Cons of using fluent.runtime

- Fluent.runtime is still young compared to gettext, therefore the community and

support aren't as extensive. - Its complexity and design may be way above what small projects need.

3.1.2 fluent.runtime

```
pip install aiogram-i18n[runtime]
```

3.1.3 Fluent compile

fluent-compiler is a Python implementation of Project Fluent, a localization framework designed to unleash the entire expressive power of natural language translations.

it is mainly used to compile FTL files to AST (Abstract Syntax Tree). It is very useful for run-time parsing.

Pros of using fluent-compiler

- It's beneficial for handling FTL files on the runtime and compiling them into AST for efficient use.

Cons of using fluent-compiler

- As with Fluent.runtime, fluent-compiler is relatively new and as such doesn't have

as extensive resources or community.

Install fluent-compiler

```
pip install aiogram-i18n[compiler]
```

3.1.4 GNU gettext

GNU gettext is an internationalization (i18n) and localization (l10n) framework commonly used in open source projects. It works by providing string function calls in your code that wraps strings meant for user interface, and then it generates .po files which are human-readable and editable, containing the actual translations.

Pros of using GNU gettext

- Gettext is widely used and has a large community. It means there are many tools,

guides and supports available for it. - Offers good support for plurals. - It handles the translation outside of your code which makes your code cleaner. - Gettext is part of the GNU Project, being present in most of the Linux Distros.

Cons of using GNU gettext

- It can be more difficult to provide context for your strings, because the only context

you can give for your source strings are comments in your code. - Its complexity can be intimidating for non-technical translators.

Install GNU gettext

```
pip install aiogram-i18n[gettext]
```

Fluent_compile

class aiogram_i18n.cores.fluent_compile_core.**FluentCompileCore**(*path: str | Path*, *default_locale: str | None = None*, *use_isolating: bool = False*, *functions: Dict[str, Callable[[...], Any]] | None = None*, *raise_key_error: bool = True*, *use_td: bool = True*, *locales_map: Dict[str, str] | None = None*)

Bases: *BaseCore*[FluentBundle]

property available_locales: Tuple[str, ...]

default_locale: str | None

find_locales() → Dict[str, FluentBundle]

Load all compiled locales from path

Returns
dict with locales

get(*message: str*, *locale: str | None = None*, */*, ***kwargs: Any*) → str

get_locale(*locale: str | None = None*) → str

get_translator(*locale: str*) → Translator

locales: Dict[str, Translator]

locales_map: Dict[str, str]

nget(*singular: str*, *plural: str | None = None*, *n: int = 1*, *locale: str | None = None*, */*, ***kwargs: Any*) → str

async shutdown() → None

async startup() → None

Fluent.runtime

class aiogram_i18n.cores.fluent_runtime_core.**FluentRuntimeCore**(*path: str | Path*, *default_locale: str | None = None*, *use_isolating: bool = False*, *functions: Dict[str, Callable[[...], Any]] | None = None*, *pre_compile: bool = True*, *raise_key_error: bool = True*, *use_td: bool = True*, *locales_map: Dict[str, str] | None = None*)

Bases: *BaseCore*[FluentBundle]

property available_locales: Tuple[str, ...]

default_locale: str | None

find_locales() → Dict[str, FluentBundle]

get(*message_id: str*, *locale: str | None = None*, */*, ***kwargs: Any*) → str

get_locale(*locale: str | None = None*) → str

get_translator(*locale: str*) → Translator

locales: Dict[str, Translator]

locales_map: Dict[str, str]

nget(*singular: str*, *plural: str | None = None*, *n: int = 1*, *locale: str | None = None*, */*, ***kwargs: Any*) → str

async shutdown() → None

async startup() → None

GNU Gettext

class aiogram_i18n.cores.gnu_text_core.**GNUTextCore**(***, *path: str | Path*, *default_locale: str | None = None*, *raise_key_error: bool = False*, *locales_map: Dict[str, str] | None = None*)

Bases: *BaseCore*[GNUTranslations]

property available_locales: Tuple[str, ...]

default_locale: str | None

find_locales() → Dict[str, GNUTranslations]

Load all compiled locales from path :return: dict with locales

get(*message: str*, *locale: str | None = None*, */*, ***kwargs: Any*) → str

get_locale(*locale: str | None = None*) → str

get_translator(*locale: str*) → Translator

locales: Dict[str, Translator]

locales_map: Dict[str, str]

nget(*singular: str*, *plural: str | None = None*, *n: int = 1*, *locale: str | None = None*, */*, ***kwargs: Any*) → str

async shutdown() → None

async startup() → None

Writing own backends

Is an abstract base class for implementing core functionality for translation.

3.2 Poem

```
In the Realm of Bots where languages intertwine,
aiogram_i18n stands tall, a beacon so divine.
Breathes life into bots, with interaction versatile,
Engaging users via tongues, in a manner user-tile.

A supportive cradle for advanced localization,
Adept in context maintenance, rid of complications.
Promises an alluring experience, quite immersive,
For every bot user, without being non-permissive.

Expand your bot's reach in the world's symphony,
With the power of aiogram_i18n's harmony.
Supports Fluent, gettext, the localization duet,
Increasing your bot's range, on that you can bet.

Choose a translation file format with adaptability,
Multilingual bots are no more a liability.
Two systems supported, the translation's a smoother sail,
In the multilingual bot creation, aiogram_i18n unveils.

Harness the strength of both, boost your bot's appeal,
Localization's no longer a task in ordeal.
Our beloved developers, lend an ear and a half,
With aiogram_i18n, give your bot users a reason to laugh.


*by JetBrains AI Assistant*
```

PYTHON MODULE INDEX

a

INDEX