surplus on wheel bridges

official bridges

there are two currently “official” bridges:

- surplus on wheels: WhatsApp Bridge
- surplus on wheels: Telegram Bridge

bring your own bridge

an informal specification

s+ow bridges are relatively simple as they are:

1. an executable or script
2. that reads in `SPOW_TARGETS` given by surplus to the bridge, using the standard input (stdin) stream
    a. bridges do not need to account for the possibility of multiple lines sent to stdin
    b. bridges should account for the possibility of comma and space (`", "` instead of just `","`) delimited targets, and strip each target of preceding and trailing whitespace
    c. bridges should recognise a platform based on a prefix (e.g. `wa:` for WhatsApp, `tg:` for Telegram, etc.)
3. reads `SPOW_MESSAGE` (`~/.cache/spow/message`) for the message content

notes:

1. stderr and stdout are redirected to s+ow’s error and output logs respectively unless the `-p / --private` flag is passed to surplus
2. any errors encountered by the bridge should always result in a non-zero return. error logs will show the exact error code, so feel free to use other numbers than 1
3. persistent data such as credentials and session data storage are to be handled by the bridge itself. consider storing them in `$HOME/.local/share/<bridge-name>/`, or wherever appropriate

example

if i were to recommend an example on a basic bridge implementation, it would be the Telegram Bridge:

src/spow-telegram-bridge/bridge.py

```python
#!/usr/bin/env python3
"""
s+ow-telegram-bridge: add-on bridge for surplus on wheels (s+ow) to telegram
----------------------------------------------------------------------------
by mark <mark@joshwel.co>

This is free and unencumbered software released into the public domain.

Anyone is free to copy, modify, publish, use, compile, sell, or
distribute this software, either in source code form or as a compiled
binary, for any purpose, commercial or non-commercial, and by any
means.

In jurisdictions that recognize copyright laws, the author or authors
of this software dedicate any and all copyright interest in the
software to the public domain. We make this dedication for the benefit
of the public at large and to the detriment of our heirs and
successors. We intend this dedication to be an overt act of
relinquishment in perpetuity of all present and future rights to this
software under copyright law.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.

For more information, please refer to <http://unlicense.org/>
"""

import asyncio
from os import environ
from pathlib import Path
from sys import argv, stderr, stdin
from traceback import print_tb
from typing import Final

from telethon import TelegramClient  # type: ignore

# exit codes:
# 1 - bad command usage or missing env vars
# 2 - bad target
# 3 - could not send message

# rundown:
# 1. if argv[-1] is 'login', then run login() and exit
# 2. read stdin and comma split it
# 3. read ~/.cache/s+ow/message
# 4. for each target in comma split stdin that starts with "tg:",
#    send ~/.cache/s+ow/message


SESSION_NAME: Final[str] = "spowtg"

dir_data: Path = Path.home().joinpath(".local/share/s+ow-telegram-bridge")
dir_data.mkdir(parents=True, exist_ok=True)

dir_cache: Path = Path.home().joinpath(".cache/s+ow-telegram-bridge")
dir_cache.mkdir(parents=True, exist_ok=True)

api_id: str | None = environ.get("SPOW_TELEGRAM_API_ID", None)
```

```python
api_hash: str | None = environ.get("SPOW_TELEGRAM_API_HASH", None)
message_file: Path = Path.home().joinpath(".cache/s+ow/message")
session_file: Path = dir_data.joinpath(f"{SESSION_NAME}.session")


def handle_error(
    exc: Exception | None = None,
    err_message: str = "error",
    recoverable: bool = False,
    exit_code: int = -1,
) -> None:
    try:
        exc_details: str = ""
        if isinstance(exc, Exception):
            exc_details = f": {exc} ({exc.__class__.__name__})"
            print_tb(exc.__traceback__, file=stderr)

        print(
            f"s+ow-telegram-bridge: {err_message}{exc_details}",
            file=stderr,
        )

    except Exception:
        pass

    if not recoverable:
        exit(exit_code)


def validate_vars() -> None:
    if api_id is None:
        print("s+ow-telegram-bridge: error: SPOW_TELEGRAM_API_ID not set", file=stderr)
        exit(1)

    if api_hash is None:
        print("s+ow-telegram-bridge: error: SPOW_TELEGRAM_API_HASH not set", file=stderr)
        exit(1)


async def run() -> None:
    silent: bool = "--silent" in argv
    delete_last: bool = "--delete-last" in argv

    if silent:
        print("s+ow-telegram-bridge: info: --silent passed", file=stderr)

    if delete_last:
        print("s+ow-telegram-bridge: info: --delete-last passed", file=stderr)

    targets: list[int] = []

    # "spec" point 2:
    #   reads in SPOW_TARGETS given by surplus to the bridge using stdin
    # "spec" point 2(a):
    #   bridges do not need to account for the possibility of multiple lines sent to stdin
    #   this bridge doesn't do this because it's simpler to iterate through stdin with a
'for'
    #   loop in python
    for line in stdin:
        for _target in line.split(","):
            # "spec" point 2(b):
            #   bridges should account for the possibility of comma and space delimited
targets
            _target = _target.strip()
            # "spec" point 2(c):
            #   bridges should recognise a platform based on a prefix
            if _target.startswith("tg:"):
                _target = _target[3:]
                if not (
                    _target.isnumeric()
                    or (_target.startswith("-") and _target.lstrip("-").isnumeric())
                ):
                    continue

                try:
                    targets.append(int(_target))

                except Exception as exc:
                    handle_error(
                        exc=exc,
                        err_message=f"error: could not cast '{_target}' as int",
                        recoverable=True,
                        exit_code=2,
                    )
                    continue

    # "spec" point 3:
    #   reads SPOW_MESSAGE (~/.cache/spow/message) for the message content
    if not (message_file.exists() and message_file.is_file()):
        print("s+ow-telegram-bridge: error: ~/.cache/s+ow/message not found", file=stderr)
        exit(1)
    message = message_file.read_text(encoding="utf-8")

    async with TelegramClient(session_file, api_id, api_hash) as client:
        for target in targets:
            try:
                if delete_last is False:
                    await client.send_message(
                        int(target),
                        message,
                        silent=silent,
                    )

                else:
                    target_persist: Path = dir_cache.joinpath(str(target))

                    try:
                        # delete old message if persist file exists
                        if target_persist.exists() and target_persist.is_file():
                            await client.delete_messages(
                                entity=target,
                                message_ids=[int(target_persist.read_text(encoding="utf-
8"))],
                            )

                    except Exception as exc:
                        handle_error(
                            exc=exc,
                            err_message="error: could not delete old message",
                            recoverable=True,
                            exit_code=3,
                        )
                        continue

                    # send new message
                    target_sent_message = await client.send_message(
                        target,
```

```python
                    message,
                    silent=silent,
                )

                # persist new message id
                target_persist.write_text(str(target_sent_message.id), encoding="utf-8")

        except Exception as exc:
            handle_error(
                exc=exc,
                err_message="error: could not send message",
                recoverable=True,
                exit_code=3,
            )
            continue

        print("s+ow-telegram-bridge: success: message sent to", target)
    exit()


def login() -> None:
    with TelegramClient(session_file, api_id, api_hash) as client:
        client.start()
    exit()


def logout() -> None:
    if session_file.exists():
        session_file.unlink()
        print("s+ow-telegram-bridge: logged out successfully", file=stderr)
    else:
        print("s+ow-telegram-bridge: already logged out", file=stderr)


def list_chats() -> None:
    with TelegramClient(session_file, api_id, api_hash) as client:
        for dialog in client.iter_dialogs():
            print(dialog.id, "\t", dialog.name)
    exit()


def entry() -> None:
    if len(argv) < 1:
        print("s+ow-telegram-bridge: error: len(argv) < 1", file=stderr)
        exit(1)

    if "login" in argv:
        validate_vars()
        login()

    elif "logout" in argv:
        logout()

    elif "list" in argv:
        validate_vars()
        list_chats()

    else:
        asyncio.run(run())


if __name__ == "__main__":
    entry()
```

Note

the feature of deleting the last sent message (`--delete-last`) is a non-standard feature for bridges, and was simply a use case i personally needed. if you're going to implement a bridge, all you really need is the ability to `login`, `logout`, and send a message

you can add other features as per the needs of your platform, like how the WhatsApp Bridge has a `pair-phone` subcommand, or per your use case needs, like in the Telegram Bridge's `--delete-last`.