```
inl api_url = "https://api.telegram.org/bot" +\ token
inl url_get_updates = api_url +\ "/getUpdates"
```

```
cargo run SpiFsxBuild -- --spi-path="./cli.spi"
"../../target/release/cli" SpiFsxBuild -- --spi-path="./cli.spi" --fsx-path="./cli.fsx"
```

```
use std::path::PathBuf;
use std::sync::mpsc::{channel, Sender};
use std::thread::{spawn, JoinHandle};
use std::time::Duration;

use notify::{RecommendedWatcher, RecursiveMode, Watcher};

#[derive(Debug)]
enum FileSystemChangeType {
    Error,
    Changed,
    Created,
    Deleted,
    Renamed,
}

#[derive(Debug)]
enum FileSystemChange {
    Error(std::io::Error),
    Changed(PathBuf),
    Created(PathBuf),
    Deleted(PathBuf),
    Renamed(PathBuf, PathBuf),
}

impl FileSystemChange {
    fn path(&self) -> Option<(PathBuf, PathBuf)> {
        match self {
            FileSystemChange::Error(_) => None,
            FileSystemChange::Changed(path)
            | FileSystemChange::Created(path)
            | FileSystemChange::Deleted(path) => (None, Some(path.clone())),
            FileSystemChange::Renamed(old_path, path) => Some((old_path.clone(), path.clone(
))),
        }
    }

    fn change_type(&self) -> FileSystemChangeType {
        match self {
            FileSystemChange::Error(_) => FileSystemChangeType::Error,
            FileSystemChange::Changed(_) => FileSystemChangeType::Changed,
            FileSystemChange::Created(_) => FileSystemChangeType::Created,
            FileSystemChange::Deleted(_) => FileSystemChangeType::Deleted,
```

```rust
            FileSystemChange::Renamed(_, _) => FileSystemChangeType::Renamed,
        }
    }
}

fn watch_with_filter(path: &str, filter: notify::RecommendedWatcher) -> Sender<FileSystemChange>
{
    let (tx, rx) = channel();

    let mut watcher: RecommendedWatcher = filter.clone();
    watcher
        .watch(path, RecursiveMode::Recursive)
        .unwrap_or_else(|e| panic!("Failed to watch directory '{}': {:?}", path, e));

    let tx2 = tx.clone();
    let mut events = watcher
        .event_receiver()
        .unwrap_or_else(|e| panic!("Failed to receive events for directory '{}': {:?}", path,
e));

    spawn(move || loop {
        match events.recv_timeout(Duration::from_secs(1)) {
            Ok(event) => match event {
                notify::DebouncedEvent::Write(path) => tx.send(FileSystemChange::Changed(
path)).unwrap(),
                notify::DebouncedEvent::Create(path) => tx.send(FileSystemChange::Created(
path)).unwrap(),
                notify::DebouncedEvent::Remove(path) => tx.send(FileSystemChange::Deleted(
path)).unwrap(),
                notify::DebouncedEvent::Rename(old_path, path) => {
                    tx.send(FileSystemChange::Renamed(old_path, path)).unwrap()
                }
                _ => {}
            },
            Err(_) => {}
        }
    });

    tx2
}

fn watch(path: &str) -> Sender<FileSystemChange> {
    watch_with_filter(
        path,
        notify::Watcher::new(
            std::time::Duration::from_secs(2),
        ).unwrap_or_else(|e| panic!("Failed to create watcher for directory '{}': {:?}",
path, e)),
    )
}
```