RITlug TeleIRC Documentation

Release 1.3.3

Mark Repka, Seth Hendrick, Nate Levesque, Robby O'Connor, Jus

Mar 27, 2020

About TeleIRC:

RITlug TeleIRC is a NodeJS implementation of a Telegram <=> IRC bridge. TeleIRC works with any IRC channel and Telegram group. It bridges messages between a Telegram group and an IRC channel.

This bot was originally written for RITlug. Today, it is used by various communities.

CHAPTER 1

Who uses TeleIRC?

The following list of projects and communities use RITlug TeleIRC:

- BSD Argentina (@bsdar | #bsdar)
- CityZen app (@CityZenApp | #cityzen)
- **Fedora Project (@fedora | #fedora-telegram)**
  - Fedora Albania (@FedoraAlbania | #fedora-sq)
  - Fedora CommOps (@fedoracommops | #fedora-commops)
  - Fedora Diversity and Inclusion Team (@fedoradiversity | #fedora-diversity)
  - Fedora LATAM (@fedoralat | #fedora-latam)
  - Fedora Marketing Team (@fedoramktg | #fedora-mktg)
  - Flock to Fedora (@flocktofedora | #fedora-flock)
- FOSS@MAGIC at RIT (@fossrit | #rit-foss)
- Hamara Linux (@hamaraLinux | #hamara)
- **LibreOffice Community (@libreofficecommunity | #libreoffice-telegram)**
  - LibreLadies (*Telegram invite-only* | #libreladies)
  - LibreOffice AppImage (*no public invite link* | #libreoffice-appimage)
  - LibreOffice Design Team (*no public invite link* | #libreoffice-design)
  - LibreOffice QA Team (*no public invite link* | #libreoffice-qa)
- Linux Users Massa Carrara (Telegram | IRC)
- MINECON Agents (*no public invite link* | #MineconAgents)
- MusicBrainz (@musicbrainz | #musicbrainz-telegram)
- Pure Data (@puredata | #dataflow)
- **RITlug (@ritlug | #rit-lug)**

  - RITlug teleirc (@teleirc | #rit-lug-teleirc)

1.1 How to get on this list

Want to have your community added to this page? Let us know you're using TeleIRC too! Submit an issue against this repo with the following info:

- Organization / group name and website
- Telegram group URL
- Your IRC channel

To be added, your group must not discuss illegal, illicit, or generally inappropriate content.

CHAPTER 2

Quick install

This is a quick installation guide for an administrator to configure and deploy TeleIRC. TeleIRC configuration is divided into these steps:

1. Create a Telegram bot
2. Configure IRC channel
3. Configure and run TeleIRC

2.1 Create a Telegram bot

Note: TeleIRC **DOES NOT** support channels, only groups. Read more about channels vs groups here.

Create a new Telegram bot to act as a bridge from the Telegram side. The bot API provides a token key for the bot. Use the bot to discover the unique chat ID of your Telegram group.

2.1.1 Create bot with BotFather

1. Start new message to @BotFather user on Telegram
2. Send `/start` to @BotFather[1]
3. Follow instructions to create new bot (e.g. name, username, description, etc.)
4. Receive **token key** for new bot (used to access Telegram API)
5. (**REQUIRED**) Set `/setprivacy` to **DISABLED** (so bot can see messages)[2]

[1] @BotFather is the Telegram bot for creating Telegram bots

[2] Privacy setting must be disabled for TeleIRC bot to see messages in the Telegram group. By default, bots cannot see messages unless a person uses a command to interact with the bot. Since TeleIRC forwards all messages, it needs to see all messages. Messages are not stored or tracked by TeleIRC.

6. Add bot to Telegram group you plan to bridge

2.1.2 Optional configuration changes

1. Set description or profile picture for your bot with @BotFather
2. Block your bot from being added to more groups (`/setjoingroups`)

2.2 Configure IRC channel

There is no required configuration for an IRC channel. However, there are recommendations for best practices:

1. Register your channel
2. **Give permanent voice to your bridge bot via ChanServ (most networks use the +V flag)**
    - *Example*: On freenode, `/query ChanServ ACCESS #channel ADD my-teleirc-bot +V`

2.3 Configure and run TeleIRC

This section explains how to configure and install TeleIRC itself.

2.3.1 Requirements

- git
- nodejs (v8 and v10 supported)
- yarn

2.3.2 Install dependencies

1. Clone the repository (`git clone https://github.com/RITlug/teleirc.git`)
2. Install dependencies (`yarn`)

2.3.3 Configuration

TeleIRC uses dotenv to manage API keys and settings. The config file you use is a `.env` file. Copy the example file to a production file to get started (`cp env.example .env`). Edit the `.env` file with your API keys and settings.

See also:

See *Config file glossary* for detailed information.

Relay Telegram picture messages via Imgur

TeleIRC retrieves picture messages via the Telegram API. By default, picture messages from Telegram are sent to IRC through Imgur. See context for why Imgur is enabled by default.

Note: By default, TeleIRC uses the generic Imgur API key. Imgur highly recommends registering each TeleIRC bot.

To add Imgur support, follow these steps:

1. Create an Imgur account
2. **Register your bot with the Imgur API**
    - Select *OAuth2 without callback* option
3. Put client ID into `.env` file

CHAPTER 3

How to deploy TeleIRC

There are several ways to deploy TeleIRC persistently. This page offers suggestions on possible deployment options.

3.1 systemd

The **recommended deployment method** is with systemd. This method requires a basic understanding of systemd unit files. Create a unique systemd service for each TeleIRC instance.

A provided systemd service file is available. Add the systemd unit file to `/usr/lib/systemd/system/` to activate it. Now, `systemctl` can be used to control your TeleIRC instance.

Note the provided file makes two assumptions:

- Using a dedicated system user (e.g. `teleirc`)
- TeleIRC config files located at `/usr/lib64/teleirc/` (i.e. files inside TeleIRC repository)

3.2 pm2

pm2 keeps NodeJS running in the background. If you run an application and it crashes, pm2 restarts the process. pm2 also restarts processes if the server reboots. Read the pm2 documentation for more information.

After pm2 is installed, run these commands to start TeleIRC:

```
cd teleirc/
pm2 start -n my-teleirc-bot teleirc.js
```

3.3 Arch Linux AUR

On ArchLinux, see teleirc-git in the AUR. The AUR package uses the systemd method to deploy TeleIRC. Place TeleIRC configuration files in the `/usr/lib/teleirc/` directory.

3.4 Docker

Docker is another way to deploy TeleIRC. Dockerfiles and images are available in `images/`.

3.4.1 Which image do I choose?

Node Alpine Linux and Fedora images are provided.

Image	File	Size
Node Alpine Linux (`node:10-alpine`)	`Dockerfile.alpine`	374 MB
Fedora latest	`Dockerfile.fedora`	569 MB

This guide uses `alpine`. If you wish to use `fedora`, replace `alpine` with `fedora`.

3.4.2 Building Docker image

You may see errors running `yarn`. You can safely ignore them. They are not fatal.

```
docker build . -f images/Dockerfile.alpine -t teleirc
docker run -d -u teleirc --name teleirc --restart always \
    -e TELEIRC_TOKEN="000000000:AAAAAAaAAa2AaAAaoAAAA-a_aaAAaAaaaAA" \
    -e IRC_CHANNEL="#channel" \
    -e IRC_BOT_NAME="teleirc" \
    -e IRC_BLACKLIST="CowSayBot,AnotherNickToIgnore" \
    -e TELEGRAM_CHAT_ID="-0000000000000" \
    teleirc
```

3.4.3 Docker Compose

Optionally, you may use docker-compose. An example compose file is provided.

Run these commands to use Docker Compose:

```
cp images/docker-compose.yml.example docker-compose.yml
cp env.example .env
docker-compose up -d teleirc
```

CHAPTER 4

Config file glossary

This page is a glossary of different settings in the `env.example` configuration file. The values shown for the settings are their defaults. This glossary is intended for advanced users.

4.1 IRC settings

`IRC_BLACKLIST=""` Comma-separated list of IRC nicks to ignore

`IRC_BOT_NAME=teleirc` IRC nickname for bot

`IRC_CHANNEL=#channel` IRC channel for bot to join

`IRC_CHANNEL_KEY=""` IRC channel key

`IRC_SEND_STICKER_EMOJI=true` Send emojis associated with a sticker to IRC (when a Telegram user sends a sticker)

`IRC_SEND_DOCUMENT=false` Send documents and files from Telegram to IRC (why is this false by default?)

`IRC_PREFIX="<"` Text displayed before Telegram name in IRC

`IRC_SUFFIX=">"` Text displayed after Telegram name in IRC

`IRC_SERVER=chat.freenode.net` IRC server to connect to

`IRC_SERVER_PASSWORD=""` IRC server password

`IRC_SERVER_PORT=6697` IRC server port

`IRC_CERT_ALLOW_SELFSIGNED=false` Allows SSL to accept SSL certificates from non-trusted CA

`IRC_CERT_ALLOW_EXPIRED=false` Allow connecting to IRC server with SSL expired cert

`IRC_NICKSERV_SERVICE=NickServ` IRC service used for authentication

`IRC_NICKSERV_PASS=""` IRC account password to complete IRC authentication

`IRC_EDITED_PREFIX="[EDIT] "` Prefix to include when a user edits a Telegram message and it is resent to IRC

IRC_MAX_MESSAGE_LENGTH=400 Maximum length of the message that can be sent to IRC. Longer messages will be split into multiple messages.

4.2 Telegram settings

TELEGRAM_CHAT_ID=-0000000000000 Telegram chat ID of bridged group (*how do I get this?*)

TELEIRC_TOKEN=000000000:AAAAAAaAAa2AaAAaoAAAA-a_aaAAaAaaaAA Private API token for Telegram bot

MAX_MESSAGES_PER_MINUTE=20 Maximum number of messages sent to Telegram from IRC per minute

SHOW_ACTION_MESSAGE=true Relay action messages (e.g. `/me thinks TeleIRC is cool!`)

SHOW_JOIN_MESSAGE=false Send Telegram message when someone joins IRC channel

SHOW_KICK_MESSAGE=true Send Telegram message when someone is kicked from IRC channel

SHOW_LEAVE_MESSAGE=false Send Telegram message when someone leaves IRC channel

4.3 Imgur settings

USE_IMGUR_FOR_IMAGES=true Upload picture messages from Telegram to Imgur, send Imgur link to IRC

IMGUR_CLIENT_ID=0000000000 Imgur API client ID value to access Imgur API

4.4 Miscellaneous settings

NTBA_FIX_319=1 Required to fix a bug in a library used by TeleIRC. For context, see yagop/node-telegram-bot-api#319.

CHAPTER 5

Frequently asked questions

This page collects frequently asked scenarios or problems with TeleIRC. Did you find something confusing? Please let us know in our developer chat or open a new pull request with a suggestion!

5.1 IRC

5.1.1 Messages do not appear in the IRC channel. Why?

There are a lot of things that *could* be the cause. However, make sure the **IRC channel is surrounded by quotes in the ‘.env‘ file**:

```
` IRC_CHANNEL="#my-cool-channel" `
```

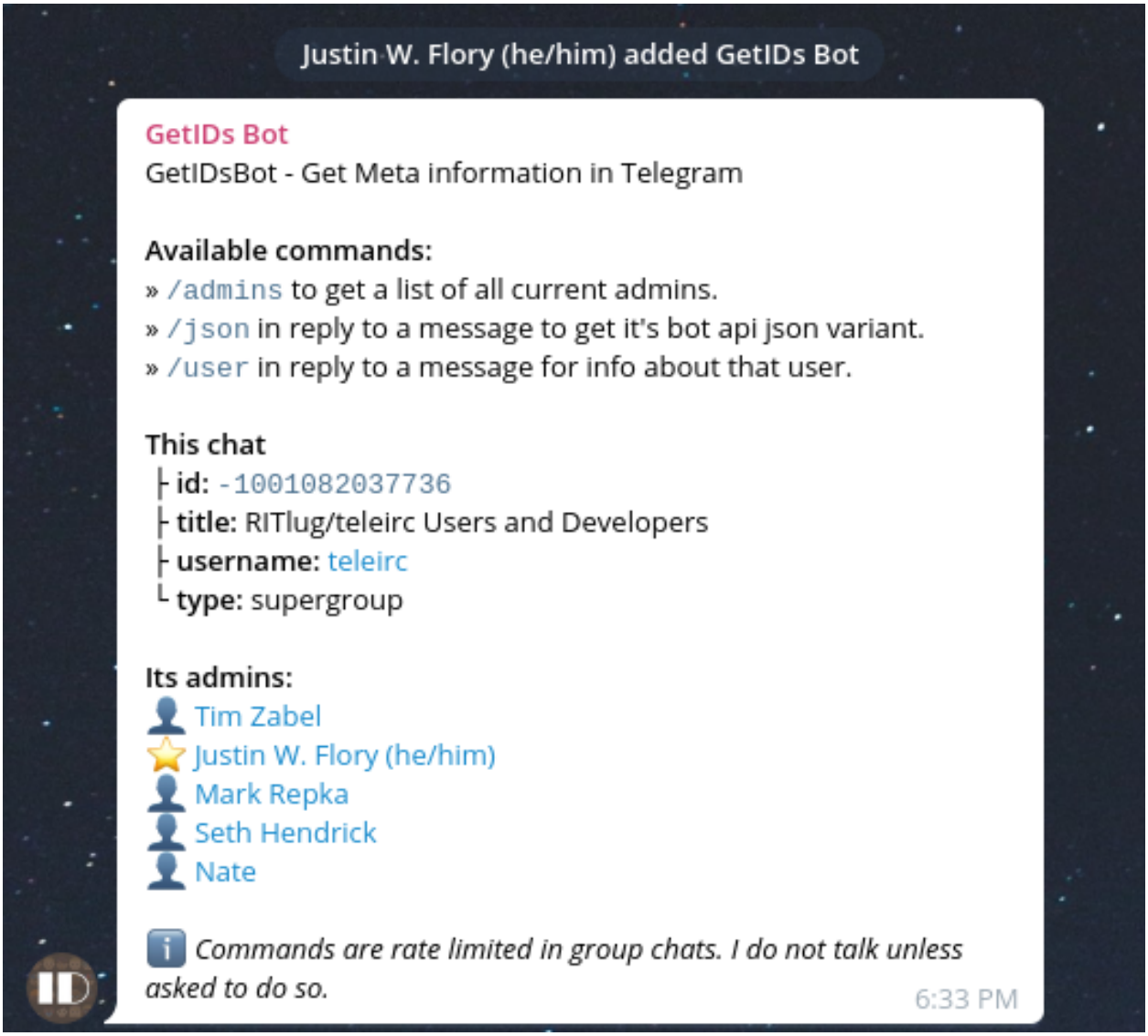
If there are no quotes, the value is interpreted as a comment, or the same thing as if it were an empty string.

5.2 Telegram

5.2.1 How do I find a chat ID for a Telegram group?

There are two ways we suggest finding the chat ID of a Telegram group.

The easiest way is to add the @getidsbot to the group. As soon as the bot joins the group, it will print a message with the group chat ID. You can remove the bot once you get the chat ID.



Another way to get the chat ID is from the Telegram API via a web browser. First, add your bot to the group. Then, open a browser and enter the Telegram API URL with your API token, as explained in this post. Next, send a message in the group that @tags the bot username, and refresh the browser window. You will see the chat ID for the Telegram group along with other information.

5.2.2 I reinstalled TeleIRC after it was inactive for a while. But the bot doesn't work. Why?

If a Telegram bot is not used for a while, it "goes to sleep". Even if TeleIRC is configured and installed correctly, you need to "wake up" the bot. To fix this, *remove the bot from the group and add it again*. Restart TeleIRC and it should work again.

CHAPTER 6

Contributing guidelines

This guide explains how to contribute to the TeleIRC project. It explicitly defines working practices of the development team. This document helps new contributors get up to speed with working on the project. It is a living document and will change. If you think something could be better, please open an issue with your feedback.

6.1 Table of contents

6.2 Set up a development environment

Contents:

6.2.1 Requirements

To set up a TeleIRC development environment, you need the following:

- Nodejs (v10+ preferred)
- Telegram account

- IRC client (HexChat recommended)
- For docs: Python 3 (3.6+ preferred)

6.2.2 Create Telegram bot

Create a Telegram bot using the Telegram BotFather. See *TeleIRC documentation* for more instructions on how to do this.

6.2.3 Create Telegram group

Create a new Telegram group for testing. Invite the bot user as another member to the group. Configure the Telegram bot to TeleIRC specifications before adding it to the group.

6.2.4 Register IRC channel

Registering an IRC channel is encouraged, but optional. At the least, you need an unused IRC channel to use for testing. Registering the channel gives you additional privileges as a channel operator (e.g. testing NickServ authentication to join private IRC channels). See your IRC network's documentation on registering a channel.

6.2.5 Configure and run TeleIRC

Change the `env.example` file to `.env`. Change the configuration values to the Telegram bot's tokens. For more help with configuration, see the *TeleIRC documentation*.

6.3 Open a new pull request

These guidelines help maintainers review new pull requests. Stick to the guidelines for quicker and easier pull request reviews.

1. Prefer gradual small changes than sudden big changes
2. Write a helpful title for your pull request (if someone reads only one sentence, will they understand your change?)
3. Address the following questions in your pull request:
    1. What is a summary of your change?
    2. Why is this change helpful?
    3. Any specific details to consider?
    4. What do you think is the outcome of this change?

6.4 Maintainer response time

Project maintainers are committed to **no more than 10 days for a reply** to a new ticket. Current maintainers are volunteers working on the project, so we try to keep up with the project as best we can. If more than 10 days passed and you have not received a reply, follow up in Telegram or IRC (`#rit-lug-teleirc` on irc.freenode.net). Someone may have missed your comment – we are not intentionally ignoring anyone.

Remember, using issue templates and answering the above questions in new pull requests likely reduces response time from a maintainer to your ticket / PR.

CHAPTER 7

Message State Diagram

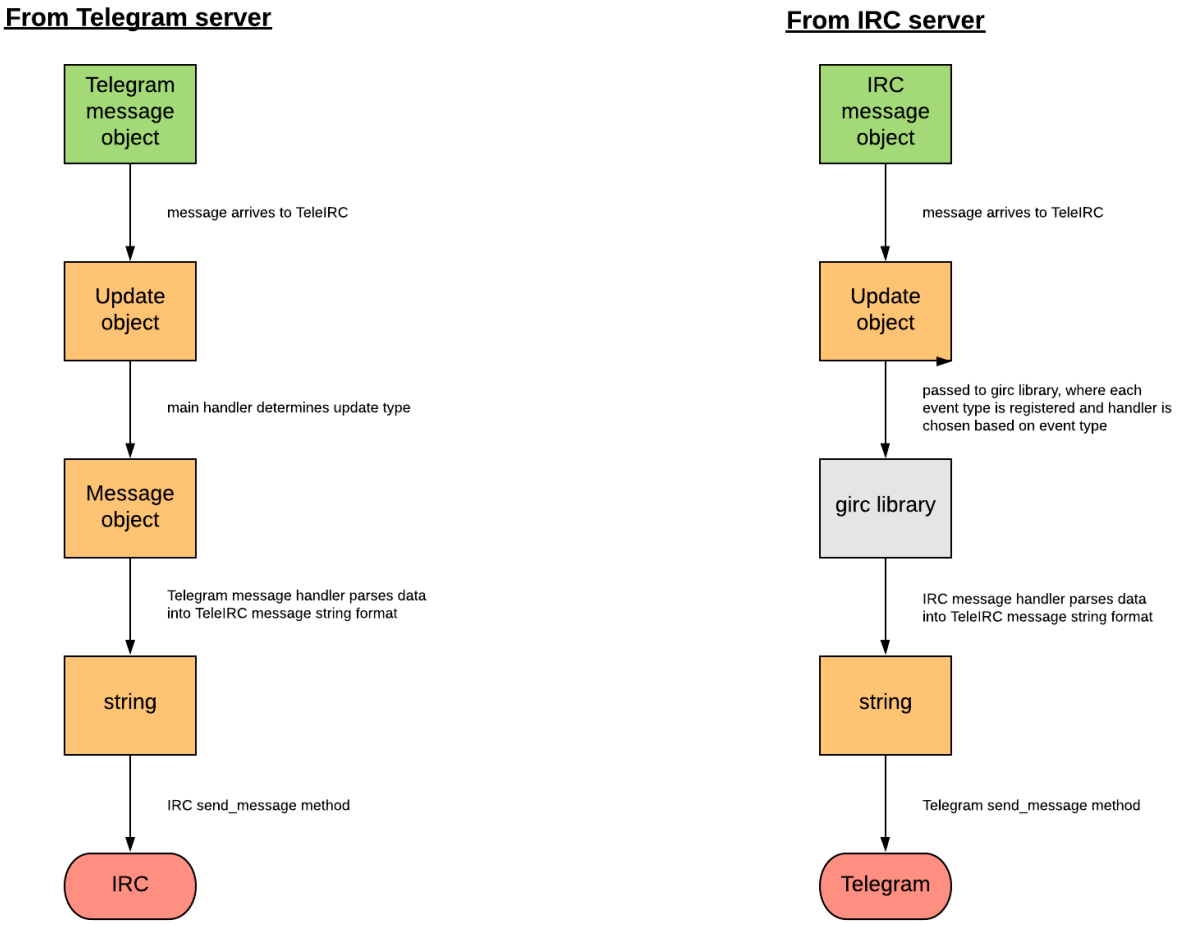
New in version v2.0.0.

Note: Only applicable to v2.0+ Golang port.

This document explains possible states of a Message as it travels through TeleIRC. There are two possible pathways, depending if a message is sourced from Telegram or IRC:

TeleIRC Message State Diagram

Prepared by Justin W. Flory
Last updated: 2020 February 15



CHAPTER 8

Go testing: Best practices

This page explains our best practices for writing unit tests in Go. This was originally written by contributor Nicholas Jones (@Zedjones).

8.1 What is unit testing?

Unit testing is a type of testing where, as the name implies, we want to test an individual *unit* of code. In any given programming language, a *unit* is the smallest piece of grouped code, usually a function.

In a unit test, we only want to test that this piece of code does its job. We don't care if any other piece of code does its job, and optimally a unit test should pass even if every other unit of code that it calls fails.

To give an example from our current code base:

```
func (c Client) StartBot(errChan chan<- error, sendMessage func(string)) {
        fmt.Println("Starting up IRC bot...")
        c.sendToTg = sendMessage
        c.addHandlers()
        if err := c.Connect(); err != nil {
                errChan <- err
        } else {
                errChan <- nil
        }
}
```

In this example, we care that our function does the following things:

- Passes "Starting up IRC bot..." to `fmt.Println`.
- Sets `c.sendToTg` to the `sendMessage` passed in.
- Calls `c.addHandlers`.
- Calls `c.Connect`
  - If an error is returned from this function, passed the error over the `errChan` channel passed in.

  - Otherwise, passes `nil` over the `errChan` channel.

Here are some things that we don't care about:

- What `fmt.Println` does with our string, that's its responsibility.
- What `c.addHandlers` does
- What `c.Connect` does, we only want its return values

In fact, we have an issue: if `c.Connect` *actually* gets called, we can't guarantee that it will ever return what we want it to so we can actually test all of our cases. Not to mention, we want our unit tests to pass even if there is no network connection available.

The solution to this: **mocking**. Due to static typing and the lack of inheritance in Go, the only real way to do proper mocking is to use interfaces.

8.2 Interfaces

An interface in Go is similar to most other languages. It defines a contract that a concrete implementation must follow.

For example:

```
type Shape interface {
    Area() float64
}
```

This interface defines what a shape is: anything that has an area. A concrete implementation would be:

```
type Rectangle struct {
    Width  float64
    Height float64
}

func (r Rectangle) Area() float64 {
    return r.Width * r.Height
}
```

Another implementation might be:

```
type Circle struct {
    Radius float64
}

func (c Circle) Area() float64 {
    return math.Pi * math.Pow(c.Radius, 2)
}
```

Now, we can create a function that accepts our interface:

```
func PrintArea (s Shape) {
    fmt.Println(s.Area())
}

func main() {
        PrintArea(Rectangle{Width: 10, Height: 15}) // prints 150
        PrintArea(Circle{Radius: 15}) // prints 706.8583
}
```

8.2.1 Why do we *need* interfaces for testing?

Because we cannot modify the implementation of functions on structs directly, we must change our functions to accept *interfaces* instead.

So, looking at one of our handlers:

```
func connectHandler(c Client) func(*girc.Client, girc.Event) {
        return func(gc *girc.Client, e girc.Event) {
                c.Cmd.Join(c.Settings.Channel)
        }
}
```

In this case, there is a small problem: we called `c.Cmd.Join`. So how does that work? How can we mock an inner struct inside of a struct? Well, we define an interface as such:

```
type IRCClient interface {
    Join(string)
}
```

Then, we modify our Client struct and add the following method:

```
func (c Client) Join(channel string) {
        c.Cmd.Join(channel)
}
```

Now, our `Client` struct implements our interface and we can change the original function to accept our interface instead and change from `c.Cmd.Join` to `c.Join`:

```
func connectHandler(c IRCClient) func(*girc.Client, girc.Event) {
        return func(gc *girc.Client, e girc.Event) {
                c.Join(c.Settings.Channel)
        }
}
```

While this does add complexity to our code, it also decouples the process of joining a channel from our exact library implementation. Now, if we want to change our IRC library down the line, we just need to write a `Join` method that fulfills this contract.

As an added bonus, we can now write our own implementation of the `IRCClient` to use as a mock. Something that fulfills the contract of the interface, but where the `Join` method doesn't actually do any work, just verifies that the string passed in is correct. **Or**, even better, let's use a library to generate this for us.

8.3 Generating mocks from interfaces with libraries

The most popular mock generation library is `mockgen`.

So, to generate a mock from the previous interface, we would run the following commands:

```
$ GO111MODULE=on go get github.com/golang/mock/mockgen@latest
$ mockgen -source=./internal/handlers/irc.go
# OR
$ mockgen github.com/ritlug/teleirc/internal/handlers/irc IRCClient
```

Now, we could write the following code in a unit test:

```
func TestConnectHandler(t *testing.T) {
  ctrl := gomock.NewController(t)

  // Assert that Join() is invoked.
  defer ctrl.Finish()

  m := NewMockIRCClient(ctrl)
  m.Settings = IRCSettings {Channel: "some channel"}

  // Asserts that the first and only call to Join() is passed "some channel".
  // Anything else will fail.
  m.
    EXPECT().
    Join(gomock.Eq("some channel"))

  connectHandler(m)(nil, nil) // Disclaimer: I didn't actually test this
}
```

CHAPTER 9

Live demo

A public Telegram supergroup and IRC channel (on Freenode) are available for testing. Our developer community is found in these channels.

- Telegram: @teleirc
- IRC: #rit-lug-teleirc (chat.freenode.net)

CHAPTER 10

Indices and tables

- github.com/RITlug/teleirc
- genindex
- modindex
- search