Reactions

Reacting to messages

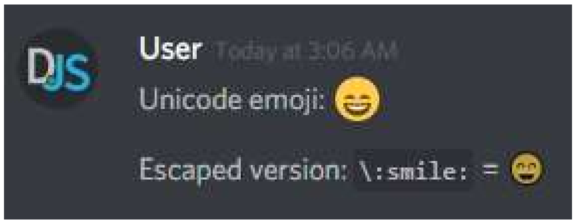
One of the first things many people want to know is how to react with emojis, both custom and "regular" (unicode). There are different routes you need to take for each of those, so let's take a look at both.

Here's the base code we'll be using:

```js
const Discord = require('discord.js');
const client = new Discord.Client();

client.once('ready', () => {
    console.log('Ready!');
});

client.on('message', message => {
    // ...
});

client.login('your-token-goes-here');
```

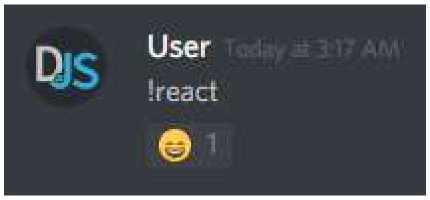
Unicode emojis

To react with a unicode emoji, you will need the actual unicode character of the emoji. There are many ways to get a unicode character of an emoji, but the easiest way would be through Discord itself. If you send a message with a unicode emoji (such as `:smile:`, for example) and put a `\` before it, it will "escape" the emoji and will display the unicode character instead of the normal emoji image.


User Today at 3:06 AM
Unicode emoji: 😄
Escaped version: `\:smile:` = 😄

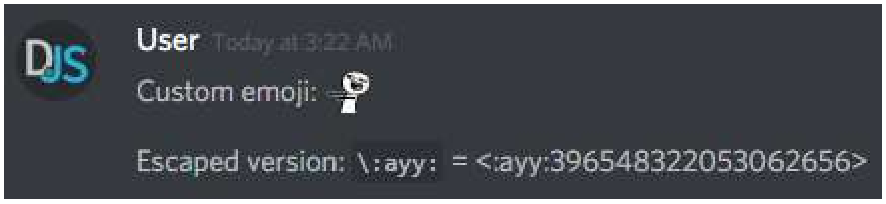
To actually react with an emoji, you need to use the `message.react()` method. Once you have the emoji character, all you need to do is copy & paste it as a string inside the `.react()` method!

```js
if (message.content === '!react') {
    message.react('😄');
}
```


User Today at 3:17 AM
!react
😄 1

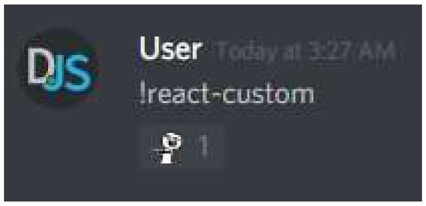
Custom emojis

For custom emojis, there are actually multiple ways of reacting. Like unicode emojis, custom emojis can also be escaped. However, when you escape a custom emoji, the result will be different.


User Today at 3:22 AM
Custom emoji:
Escaped version: `\:ayy:` = <:ayy:396548322053062656>

This format is essentially the name of the emoji, followed by its ID. Copy & paste the ID into the `.react()` method as a string.

```js
if (message.content === '!react-custom') {
    message.react('396548322053062656');
}
```


User Today at 3:27 AM
!react-custom

Great! This route may not always be available to you, though. Sometimes you'll need to react with an emoji programmatically. To do so, you'll need to retrieve the emoji object.

Two of the easiest ways you can retrieve an emoji would be:

- Use `.find()` on a Collection of Emojis.
- Use `.get()` on the `client.emojis.cache` Collection.

> **TIP**
>
> It is possible for two or more emojis to have the same name, and using `.find()` will only return the **first** entry it finds. As such, this can cause unexpected results.

Using `.find()`, your code would look something like this:

```js
if (message.content === '!react-custom') {
	const reactionEmoji = message.guild.emojis.cache.find(emoji => emoji.name === 'ayy'
	message.react(reactionEmoji);
}
```

Using `.get()`, your code would look something like this:

```js
if (message.content === '!react-custom') {
	const reactionEmoji = client.emojis.cache.get(config.emojiID);
	message.react(reactionEmoji);
}
```

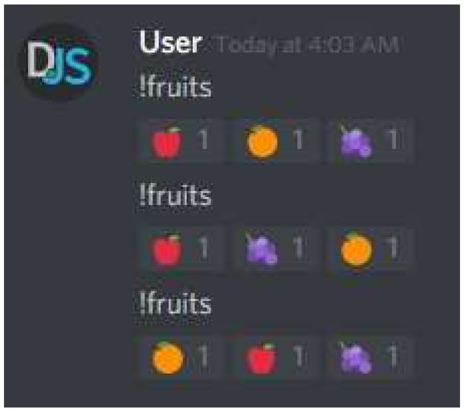
Of course, if you already have the emoji ID, you should just put that directly inside the `.react()` method. But if you want to do other things with the emoji data later on (e.g. display the name or image URL), it's best to retrieve the full emoji object.

Reacting in order

If you just put one `message.react()` under another, it won't always react in order as is. This is because `.react()` is a Promise, and as such, an asynchronous operation.

js

```js
if (message.content === '!fruits') {
    message.react('🍎');
    message.react('🍊');
    message.react('🍇');
}
```



User Today at 4:03 AM

!fruits

🍎 1 🍊 1 🍇 1

!fruits

🍎 1 🍇 1 🍊 1

!fruits

🍊 1 🍎 1 🍇 1

As you can see, if you leave it like that, it won't display as you really want it to. It was able to react correctly on the first try, but reacts in a different order each time after that.

Luckily, there are two easy solutions to this. The first would be to to chain `.then()` s in the order you want it to display.

js

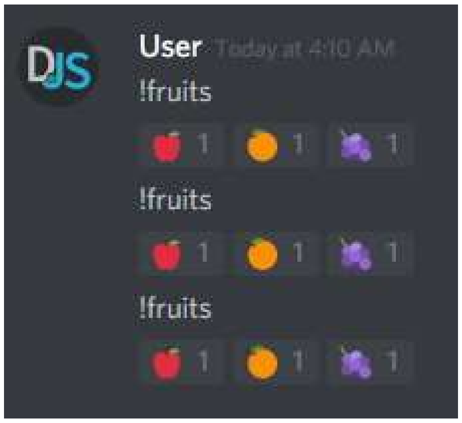
```js
client.on('message', message => {
    if (message.content === '!fruits') {
        message.react('🍎')
            .then(() => message.react('🍊'))
            .then(() => message.react('🍇'))
            .catch(() => console.error('One of the emojis failed to react.'));
    }
});
```

The other would be to use the `async` / `await` keywords.

js

```js
// notice the `async` keyword
client.on('message', async message => {
    if (message.content === '!fruits') {
        try {
            await message.react('🍎');
            await message.react('🍊');
            await message.react('🍇');
        } catch (error) {
            console.error('One of the emojis failed to react.');
        }
    }
});
```

If you try again with either of the code blocks above, you'll get the result you originally wanted!


User Today at 4:10 AM
!fruits
🍎 1 🍊 1 🍇 1
!fruits
🍎 1 🍊 1 🍇 1
!fruits
🍎 1 🍊 1 🍇 1

TIP

If you aren't familiar with Promises or `async` / `await` , you can read more about them on **MDN** or **our guide page on async/await**!

Handling multiple reactions if the order doesn't matter

However, if you don't mind the order the emojis react in, you can take advantage of `Promise.all()` , like so:

js

```js
if (message.content === '!fruits') {
    Promise.all([
        message.react('🍎'),
        message.react('🍊'),
			message.react('🍇'),
		])
			.catch(() => console.error('One of the emojis failed to react.'));
	}
```

The benefit of this small optimization is that you can use `.then()` to handle when all of the Promises have resolved, or `.catch()` when one of them has failed. You can also `await` it since it returns a Promise itself.

Removing reactions

Now that you know how to add reactions, you might be asking, how do you remove them? In this section you will learn how to remove all reactions, remove reactions by user, and remove reactions by emoji.

WARNING

All of these methods require `MANAGE_MESSAGES` permissions. Make sure your bot has permissions before attempting to utilize any of these methods, as it will error if it doesn't.

Removing all reactions

Removing all reactions from a message is the easiest, the API allows you to do this through a single call. It can be done through the `message.reactions.removeAll()` method.

```js
message.reactions.removeAll().catch(error => console.error('Failed to clear reactions:
```

Removing reactions by emoji

Removing reactions by emoji is easily done by using `MessageReaction.remove()`.

```js
message.reactions.cache.get('484535447171760141').remove().catch(error => console.error
```

Removing reactions by user

TIP

If you are not familiar with `Collection.filter()` and `Collection.has()` take the time to understand what they do and then come back.

Removing reactions by user is not as straightforward as removing by emoji or removing all reactions. The API does not provide a method for selectively removing reactions of a user. This means you will have to iterate through reactions which include the user and remove them.

```js
const userReactions = message.reactions.cache.filter(reaction => reaction.users.cache.ha
try {
    for (const reaction of userReactions.values()) {
        await reaction.users.remove(userId);
    }
} catch (error) {
    console.error('Failed to remove reactions.');
}
```

WARNING

Make sure not to remove reactions by user too much, if there are a lot of reactions or a lot of users it can be considered API spam.

Awaiting reactions

A common use case for reactions in commands is having a user confirm or deny an action, or creating a poll system. Luckily, we actually **already have a guide page that covers this**! Check out that page if you want a more in-depth explanation. Otherwise, here's a basic example for reference:

```js
message.react('👍').then(() => message.react('👎'));

const filter = (reaction, user) => {
    return ['👍', '👎'].includes(reaction.emoji.name) && user.id === message.author.id;
};

message.awaitReactions(filter, { max: 1, time: 60000, errors: ['time'] })
    .then(collected => {
        const reaction = collected.first();

        if (reaction.emoji.name === '👍') {
            message.reply('you reacted with a thumbs up.');
        } else {
            message.reply('you reacted with a thumbs down.');
        }
    })
    .catch(collected => {
        message.reply('you reacted with neither a thumbs up, nor a thumbs down.');
    });
```

Listening for reactions on old messages

Messages sent before your bot started are uncached, unless you fetch them first. By default the library does not emit client events if the data received and cached is not sufficient to build fully functional objects. Since version 12 you can change this behavior by activating partials. For a full explanation of partials see **this page**.

Make sure you enable partial structures for `MESSAGE` , `CHANNEL` and `REACTION` when instantiating your client, if you want reaction events on uncached messages for both server and direct message channels. If you do not want to support direct message channels you can exclude `CHANNEL` .

> **TIP**
>
> If you use **gateway intents** but can't or don't want to use the privileged `GUILD_PRESENCES` intent you additionally need the `USER` partial.

```js
const Discord = require('discord.js');
const client = new Discord.Client({ partials: ['MESSAGE', 'CHANNEL', 'REACTION'] });
client.on('messageReactionAdd', async (reaction, user) => {
    // When we receive a reaction we check if the reaction is partial or not
    if (reaction.partial) {
        // If the message this reaction belongs to was removed the fetching might resul
        try {
            await reaction.fetch();
        } catch (error) {
            console.error('Something went wrong when fetching the message: ', error);
            // Return as `reaction.message.author` may be undefined/null
            return;
        }
    }
    // Now the message has been cached and is fully available
    console.log(`${reaction.message.author}'s message "${reaction.message.content}" gai
    // The reaction is now also fully available and the properties will be reflected ac
    console.log(`${reaction.count} user(s) have given the same reaction to this message
});
```

WARNING

Partial structures are enabled globally. You can not only make them work for a certain event or cache and you very likely need to adapt other parts of your code that are accessing data from the relevant caches. All caches holding the respective structure type might return partials as well!

Resulting code

If you want to compare your code to the code we've constructed so far, you can review it over on the GitHub repository **here**.

Edit this page

Last Updated: 10/11/2020, 11:51:19 AM

← Permissions (extended)

Collectors →