

[Pro](#) [Teams](#) [Pricing](#) [Documentation](#)[Sign Up](#)[Sign In](#)[Search](#)**baileys** TS6.7.16 • Public • Published a month ago[Readme](#)[Code](#) Beta[14 Dependencies](#)[15 Dependents](#)[42 Versions](#)

Baileys - Typescript/Javascript WhatsApp Web API

downloads **1.9k** npm **90k/week** code size **8.2 MiB** license **MIT** discord **807 online** [Stars](#) **5.1k**
[Forks](#) **1.7k**

Important Note

This library was originally a project for **CS-2362 at Ashoka University** and is in no way affiliated with or endorsed by WhatsApp. Use at your own discretion. Do not spam people with this. We discourage any stalkerware, bulk or automated messaging usage.

Liability and License Notice

Baileys and its maintainers cannot be held liable for misuse of this application, as stated in the **MIT license**. The maintainers of Baileys do not in any way condone the use of this application in practices that violate the Terms of Service of WhatsApp. The maintainers of this application call upon the personal responsibility of its users to use this application in a fair way, as it is intended to be used.

- Baileys does not require Selenium or any other browser to be interface with WhatsApp Web, it does so directly using a **WebSocket**.
- Not running Selenium or Chromimum saves you like **half a gig** of ram :/
- Baileys supports interacting with the multi-device & web versions of WhatsApp.
- Thank you to [@pokearaujo](#) for writing his observations on the workings of WhatsApp Multi-Device. Also, thank you to [@Sigalor](#) for writing his observations on the workings of WhatsApp Web and thanks to [@Rhymen](#) for the **go** implementation.

[!IMPORTANT] The original repository had to be removed by the original author - we now continue development in this repository here. This is the only official repository and is maintained by the community. **Join the Discord [here](#)**

Example

Do check out & run **example.ts** to see an example usage of the library. The script covers most common use cases. To run the example script, download or clone the repo and then type the following in a terminal:

1. `cd path/to/Baileys`
2. `yarn`
3. `yarn example`

Install

Use the stable version:

```
yarn add baileys
```

Use the edge version (no guarantee of stability, but latest fixes + features)

```
yarn add github:WhiskeySockets/Baileys
```

Then import the default function in your code:

```
import makeWASocket from 'baileys'
```

Links

- [Discord](#)
- [Docs](#)

Index

- [Connecting Account](#)
 - [Connect with QR-CODE](#)
 - [Connect with Pairing Code](#)
 - [Receive Full History](#)
- [Important Notes About Socket Config](#)
 - [Caching Group Metadata \(Recommended\)](#)
 - [Improve Retry System & Decrypt Poll Votes](#)
 - [Receive Notifications in Whatsapp App](#)
- [Save Auth Info](#)
- [Handling Events](#)
 - [Example to Start](#)
 - [Decrypt Poll Votes](#)
 - [Summary of Events on First Connection](#)
- [Implementing a Data Store](#)
- [Whatsapp IDs Explain](#)
- [Utility Functions](#)
- [Sending Messages](#)
 - [Non-Media Messages](#)
 - [Text Message](#)
 - [Quote Message](#)
 - [Mention User](#)
 - [Forward Messages](#)
 - [Location Message](#)

- Contact Message
- Reaction Message
- Pin Message
- Poll Message
- Sending with Link Preview
- Media Messages
 - Gif Message
 - Video Message
 - Audio Message
 - Image Message
 - ViewOnce Message
- Modify Messages
 - Delete Messages (for everyone)
 - Edit Messages
- Manipulating Media Messages
 - Thumbnail in Media Messages
 - Downloading Media Messages
 - Re-upload Media Message to Whatsapp
- Reject Call
- Send States in Chat
 - Reading Messages
 - Update Presence
- Modifying Chats
 - Archive a Chat
 - Mute/Unmute a Chat
 - Mark a Chat Read/Unread
 - Delete a Message for Me
 - Delete a Chat
 - Star/Unstar a Message
 - Disappearing Messages
- User Querys

- Check If ID Exists in Whatsapp
- Query Chat History (groups too)
- Fetch Status
- Fetch Profile Picture (groups too)
- Fetch Bussines Profile (such as description or category)
- Fetch Someone's Presence (if they're typing or online)
- Change Profile
 - Change Profile Status
 - Change Profile Name
 - Change Display Picture (groups too)
 - Remove display picture (groups too)
- Groups
 - Create a Group
 - Add/Remove or Demote/Promote
 - Change Subject (name)
 - Change Description
 - Change Settings
 - Leave a Group
 - Get Invite Code
 - Revoke Invite Code
 - Join Using Invitation Code
 - Get Group Info by Invite Code
 - Query Metadata (participants, name, description...)
 - Join using groupInviteMessage
 - Get Request Join List
 - Approve/Reject Request Join
 - Get All Participating Groups Metadata
 - Toggle Ephemeral
 - Change Add Mode
- Privacy
 - Block/Unblock User
 - Get Privacy Settings

- [Get BlockList](#)
- [Update LastSeen Privacy](#)
- [Update Online Privacy](#)
- [Update Profile Picture Privacy](#)
- [Update Status Privacy](#)
- [Update Read Receipts Privacy](#)
- [Update Groups Add Privacy](#)
- [Update Default Disappearing Mode](#)
- [Broadcast Lists & Stories](#)
 - [Send Broadcast & Stories](#)
 - [Query a Broadcast List's Recipients & Name](#)
- [Writing Custom Functionality](#)
 - [Enabling Debug Level in Baileys Logs](#)
 - [How Whatsapp Communicate With Us](#)
 - [Register a Callback for Websocket Events](#)

Connecting Account

WhatsApp provides a multi-device API that allows Baileys to be authenticated as a second WhatsApp client by scanning a **QR code** or **Pairing Code** with WhatsApp on your phone.

[!NOTE] [Here](#) is a simple example of event handling

[!TIP] You can see all supported socket configs [here](#) (Recommended)

Starting socket with QR-CODE

[!TIP] You can customize browser name if you connect with **QR-CODE**, with `Browser` constant, we have some browsers config, [see here](#)

```
import makeWASocket from 'baileys'
```

```
const sock = makeWASocket({  
  // can provide additional config here
```

```

    browser: Browsers.ubuntu('My App'),
    printQRInTerminal: true
  })

```

If the connection is successful, you will see a QR code printed on your terminal screen, scan it with WhatsApp on your phone and you'll be logged in!

Starting socket with Pairing Code

[!IMPORTANT] Pairing Code isn't Mobile API, it's a method to connect Whatsapp Web without QR-CODE, you can connect only with one device, see [here](#)

The phone number can't have + or () or - , only numbers, you must provide country code

```

import makeWASocket from 'baileys'

const sock = makeWASocket({
  // can provide additional config here
  printQRInTerminal: false //need to be false
})
// NOTE: WAIT TILL QR EVENT BEFORE REQUESTING THE PAIRING CODE
if (!sock.authState.creds.registered) {
  const number = 'XXXXXXXXXXXX'
  const code = await sock.requestPairingCode(number)
  console.log(code)
}

```

Receive Full History

1. Set syncFullHistory as true
2. Baileys, by default, use chrome browser config
 - If you'd like to emulate a desktop connection (and receive more message history), this browser setting to your Socket config:

```

const sock = makeWASocket({
  ...otherOpts,

```

```
// can use Windows, Ubuntu here too
browser: Browsers.macos('Desktop'),
syncFullHistory: true
})
```

Important Notes About Socket Config

Caching Group Metadata (Recommended)

- If you use baileys for groups, we recommend you to set `cachedGroupMetadata` in socket config, you need to implement a cache like this:

```
const groupCache = new NodeCache({stdTTL: 5 * 60, useClones: false})

const sock = makeWASocket({
  cachedGroupMetadata: async (jid) => groupCache.get(jid)
})

sock.ev.on('groups.update', async ([event]) => {
  const metadata = await sock.groupMetadata(event.id)
  groupCache.set(event.id, metadata)
})

sock.ev.on('group-participants.update', async (event) => {
  const metadata = await sock.groupMetadata(event.id)
  groupCache.set(event.id, metadata)
})
```

Improve Retry System & Decrypt Poll Votes

- If you want to improve sending message, retrying when error occurs and decrypt poll votes, you need to have a store and set `getMessage` config in socket like this:

```
const sock = makeWASocket({
  getMessage: async (key) => await getMessageFromStore(key)
})
```

Receive Notifications in Whatsapp App

- If you want to receive notifications in whatsapp app, set `markOnlineOnConnect` to `false`


```
const sock = makeWASocket({  
  markOnlineOnConnect: false  
})
```

Saving & Restoring Sessions

You obviously don't want to keep scanning the QR code every time you want to connect.

So, you can load the credentials to log back in:

```
import makeWASocket, { useMultiFileAuthState } from 'baileys'  
  
const { state, saveCreds } = await useMultiFileAuthState('auth_info_  
  
// will use the given state to connect  
// so if valid credentials are available -- it'll connect without QR  
const sock = makeWASocket({ auth: state })  
  
// this will be called as soon as the credentials are updated  
sock.ev.on('creds.update', saveCreds)
```

[!IMPORTANT] `useMultiFileAuthState` is a utility function to help save the auth state in a single folder, this function serves as a good guide to help write auth & key states for SQL/no-SQL databases, which I would recommend in any production grade system.

[!NOTE] When a message is received/sent, due to signal sessions needing updating, the auth keys (`authState.keys`) will update. Whenever that happens, you must save the updated keys (`authState.keys.set()` is called). Not doing so will prevent your messages from reaching the recipient & cause other unexpected consequences. The `useMultiFileAuthState` function automatically takes care of that, but for any other serious implementation -- you will need to be very careful with the key state management.

Handling Events

- Baileys uses the EventEmitter syntax for events. They're all nicely typed up, so you shouldn't have any issues with an Intellisense editor like VS Code.

[!IMPORTANT] The events are **these**, it's important you see all events

You can listen to these events like this:

```
const sock = makeWASocket()
sock.ev.on('messages.upsert', ({ messages }) => {
  console.log('got messages', messages)
})
```

Example to Start

[!NOTE] This example includes basic auth storage too

```
import makeWASocket, { DisconnectReason, useMultiFileAuthState } from 'whatsapp-web.js'
import { Boom } from '@hapi/boom'

async function connectToWhatsApp () {
  const { state, saveCreds } = await useMultiFileAuthState('auth_data')
  const sock = makeWASocket({
    // can provide additional config here
    auth: state,
    printQRInTerminal: true
  })
  sock.ev.on('connection.update', (update) => {
    const { connection, lastDisconnect } = update
    if(connection === 'close') {
      const shouldReconnect = (lastDisconnect.error as Boom)?.isBoom()
      console.log('connection closed due to ', lastDisconnect.error)
      // reconnect if not logged out
      if(shouldReconnect) {
        connectToWhatsApp()
      }
    }
  })
}
```

```

    }
    } else if(connection === 'open') {
        console.log('opened connection')
    }
})
sock.ev.on('messages.upsert', event => {
    for (const m of event.messages) {
        console.log(JSON.stringify(m, undefined, 2))

        console.log('replying to', m.key.remoteJid)
        await sock.sendMessage(m.key.remoteJid!, { text: 'Hello'
    }
})

// to storage creds (session info) when it updates
sock.ev.on('creds.update', saveCreds)
}
// run in main file
connectToWhatsApp()

```

[!IMPORTANT] In `messages.upsert` it's recommended to use a loop like `for (const message of event.messages)` to handle all messages in array

Decrypt Poll Votes

- By default poll votes are encrypted and handled in `messages.update`
- That's a simple example

```

sock.ev.on('messages.update', event => {
    for(const { key, update } of event) {
        if(update.pollUpdates) {
            const pollCreation = await getMessage(key)
            if(pollCreation) {
                console.log(
                    'got poll update, aggregation: ',

```

```

        getAggregateVotesInPollMessage({
            message: pollCreation,
            pollUpdates: update.pollUpdates,
        })
    )
}
}
}
})

```

- `getMessage` is a **store** implementation (in your end)

Summary of Events on First Connection

1. When you connect first time, `connection.update` will be fired requesting you to restart sock
2. Then, history messages will be received in `messaging-history.set`

Implementing a Data Store

- Baileys does not come with a defacto storage for chats, contacts, or messages. However, a simple in-memory implementation has been provided. The store listens for chat updates, new messages, message updates, etc., to always have an up-to-date version of the data.

[!IMPORTANT] I highly recommend building your own data store, as storing someone's entire chat history in memory is a terrible waste of RAM.

It can be used as follows:

```

import makeWASocket, { makeInMemoryStore } from 'baileys'
// the store maintains the data of the WA connection in memory
// can be written out to a file & read from it
const store = makeInMemoryStore({ })
// can be read from a file
store.readFromFile('./baileys_store.json')
// saves the state to a file every 10s
setInterval(() => {

```

```

    store.writeToFile('./baileys_store.json')
  }, 10_000)

const sock = makeWASocket({ })
// will listen from this socket
// the store can listen from a new socket once the current socket on
store.bind(sock.ev)

sock.ev.on('chats.upsert', () => {
  // can use 'store.chats' however you want, even after the socket
  // 'chats' => a KeyedDB instance
  console.log('got chats', store.chats.all())
})

sock.ev.on('contacts.upsert', () => {
  console.log('got contacts', Object.values(store.contacts))
})

```

The store also provides some simple functions such as `loadMessages` that utilize the store to speed up data retrieval.

Whatsapp IDs Explain

- `id` is the WhatsApp ID, called `jid` too, of the person or group you're sending the message to.
 - It must be in the format `[country code][phone number]@s.whatsapp.net`
 - Example for people: `+19999999999@s.whatsapp.net` .
 - For groups, it must be in the format `123456789-123345@g.us` .
 - For broadcast lists, it's `[timestamp of creation]@broadcast` .
 - For stories, the ID is `status@broadcast` .

Utility Functions

- `getContentType` , returns the content type for any message
- `getDevice` , returns the device from message

- `makeCacheableSignalKeyStore` , make auth store more fast
- `downloadContentFromMessage` , download content from any message

Sending Messages

- Send all types of messages with a single function
 - [Here](#) you can see all message contents supported, like text message
 - [Here](#) you can see all options supported, like quote message

```
const jid: string
const content: AnyMessageContent
const options: MiscMessageGenerationOptions

sock.sendMessage(jid, content, options)
```

Non-Media Messages

Text Message

```
await sock.sendMessage(jid, { text: 'hello word' })
```

Quote Message (works with all types)

```
await sock.sendMessage(jid, { text: 'hello word' }, { quoted: message })
```



Mention User (works with most types)

- @number is to mention in text, it's optional

```
await sock.sendMessage(
  jid,
  {
    text: '@12345678901',
    mentions: ['12345678901@s.whatsapp.net']
  }
)
```

Forward Messages

- You need to have message object, can be retrieved from **store** or use a **message** object

```
const msg = getMessageFromStore() // implement this on your end
await sock.sendMessage(jid, { forward: msg }) // WA forward the mes:
```



Location Message

```
await sock.sendMessage(
  jid,
  {
    location: {
      degreesLatitude: 24.121231,
      degreesLongitude: 55.1121221
    }
  }
)
```

Contact Message

```
const vcard = 'BEGIN:VCARD\n' // metadata of the contact card
  + 'VERSION:3.0\n'
  + 'FN:Jeff Singh\n' // full name
  + 'ORG:Ashoka Uni;\n' // the organization of the contact
  + 'TEL;type=CELL;type=VOICE;waid=911234567890:+91 12345'
  + 'END:VCARD'

await sock.sendMessage(
  id,
  {
    contacts: {
      displayName: 'Jeff',
      contacts: [{ vcard }]
    }
  }
)
```

```
    }
  )
}
```

Reaction Message

- You need to pass the key of message, you can retrieve from **store** or use a **key** object

```
await sock.sendMessage(
  jid,
  {
    react: {
      text: '💖', // use an empty string to remove the reacti
      key: message.key
    }
  }
)
```

Pin Message

- You need to pass the key of message, you can retrieve from **store** or use a **key** object
- Time can be:

Time	Seconds
24h	86.400
7d	604.800
30d	2.592.000

```
await sock.sendMessage(
  jid,
  {
    pin: {
      type: 1, // 0 to remove
      time: 86400
    }
  }
)
```



```

        key: message.key
      }
    }
  )

```

Poll Message

```

await sock.sendMessage(
  jid,
  {
    poll: {
      name: 'My Poll',
      values: ['Option 1', 'Option 2', ...],
      selectableCount: 1,
      toAnnouncementGroup: false // or true
    }
  }
)

```

Sending Messages with Link Previews

1. By default, wa does not have link generation when sent from the web
2. Baileys has a function to generate the content for these link previews
3. To enable this function's usage, add `link-preview-js` as a dependency to your project with `yarn add link-preview-js`
4. Send a link:

```

await sock.sendMessage(
  jid,
  {
    text: 'Hi, this was sent using https://github.com/whiskeyso
  }
)

```

Media Messages

Sending media (video, stickers, images) is easier & more efficient than ever.

[!NOTE] In media messages, you can pass `{ stream: Stream }` or `{ url: Url }` or `Buffer` directly, you can see more [here](#)

- When specifying a media url, Baileys never loads the entire buffer into memory; it even encrypts the media as a readable stream.

[!TIP] It's recommended to use `Stream` or `Url` to save memory

Gif Message

- Whatsapp doesn't support `.gif` files, that's why we send gifs as common `.mp4` video with `gifPlayback` flag

```
await sock.sendMessage(
  jid,
  {
    video: fs.readFileSync('Media/ma_gif.mp4'),
    caption: 'hello word',
    gifPlayback: true
  }
)
```

Video Message

```
await sock.sendMessage(
  id,
  {
    video: {
      url: './Media/ma_gif.mp4'
    },
    caption: 'hello word',
    ptt: false // if set to true, will send as a `video note`
  }
)
```

Audio Message

- To audio message work in all devices you need to convert with some tool like `ffmpeg` with this flags:

```
codec: libopus //ogg file
ac: 1 //one channel
avoid_negative_ts
make_zero
```

- Example:

```
ffmpeg -i input.mp4 -avoid_negative_ts make_zero -ac 1 output.ogg
```



```
await sock.sendMessage(
  jid,
  {
    audio: {
      url: './Media/audio.mp3'
    },
    mimetype: 'audio/mp4'
  }
)
```

Image Message

```
await sock.sendMessage(
  id,
  {
    image: {
      url: './Media/ma_img.png'
    },
    caption: 'hello word'
  }
)
```

View Once Message

- You can send all messages above as `viewOnce` , you only need to pass `viewOnce: true` in content object

```
await sock.sendMessage(
  id,
  {
    image: {
      url: './Media/ma_img.png'
    },
    viewOnce: true, //works with video, audio too
    caption: 'hello word'
  }
)
```

Modify Messages

Deleting Messages (for everyone)

```
const msg = await sock.sendMessage(jid, { text: 'hello word' })
await sock.sendMessage(jid, { delete: msg.key })
```

Note: deleting for oneself is supported via `chatModify` , see in [this section](#)

Editing Messages

- You can pass all editable contents here

```
await sock.sendMessage(jid, {
  text: 'updated text goes here',
  edit: response.key,
});
```

Manipulating Media Messages

Thumbnail in Media Messages

- For media messages, the thumbnail can be generated automatically for images & stickers provided you add `jimp` or `sharp` as a dependency in your project using

yarn add jimp or yarn add sharp.

- Thumbnails for videos can also be generated automatically, though, you need to have ffmpeg installed on your system.

Downloading Media Messages

If you want to save the media you received

```
import { createWriteStream } from 'fs'
import { downloadMediaMessage, getContentType } from 'baileys'

sock.ev.on('messages.upsert', async ({ [m] }) => {
  if (!m.message) return // if there is no text or media message
  const messageType = getContentType(m) // get what type of message

  // if the message is an image
  if (messageType === 'imageMessage') {
    // download the message
    const stream = await downloadMediaMessage(
      m,
      'stream', // can be 'buffer' too
      { },
      {
        logger,
        // pass this so that baileys can request a reupload
        // that has been deleted
        reuploadRequest: sock.updateMediaMessage
      }
    )
    // save to file
    const writeStream = createWriteStream('./my-download.jpeg')
    stream.pipe(writeStream)
  }
})
```

Re-upload Media Message to Whatsapp

- WhatsApp automatically removes old media from their servers. For the device to access said media -- a re-upload is required by another device that has it. This can be accomplished using:

```
await sock.updateMediaMessage(msg)
```

Reject Call

- You can obtain `callId` and `callFrom` from `call` event

```
await sock.rejectCall(callId, callFrom)
```

Send States in Chat

Reading Messages

- A set of message **keys** must be explicitly marked read now.
- You cannot mark an entire 'chat' read as it were with Baileys Web. This means you have to keep track of unread messages.

```
const key: WAMessageKey  
// can pass multiple keys to read multiple messages as well  
await sock.readMessages([key])
```

The message ID is the unique identifier of the message that you are marking as read. On a `WAMessage`, the `messageID` can be accessed using `messageID = message.key.id`.

Update Presence

- `presence` can be one of **these**
- The presence expires after about 10 seconds.
- This lets the person/group with `jid` know whether you're online, offline, typing etc.

```
await sock.sendPresenceUpdate('available', jid)
```

[!NOTE] If a desktop client is active, WA doesn't send push notifications to the device. If you would like to receive said notifications -- mark your Baileys client

```
offline using sock.sendPresenceUpdate('unavailable')
```

Modifying Chats

WA uses an encrypted form of communication to send chat/app updates. This has been implemented mostly and you can send the following updates:

[!IMPORTANT] If you mess up one of your updates, WA can log you out of all your devices and you'll have to log in again.

Archive a Chat

```
const lastMsgInChat = await getLastMessageInChat(jid) // implement 1
await sock.chatModify({ archive: true, lastMessages: [lastMsgInChat]
```

Mute/Unmute a Chat

- Supported times:

Time	Milliseconds
Remove	null
8h	86.400.000
7d	604.800.000

```
// mute for 8 hours
```

```
await sock.chatModify({ mute: 8 * 60 * 60 * 1000 }, jid)
```

```
// unmute
```

```
await sock.chatModify({ mute: null }, jid)
```

Mark a Chat Read/Unread

```
const lastMsgInChat = await getLastMessageInChat(jid) // implement 1
// mark it unread
```

```
await sock.chatModify({ markRead: false, lastMessages: [lastMsgInChat]
```

Delete a Message for Me

```
await sock.chatModify(
  {
    clear: {
      messages: [
        {
          id: 'ATWYHDNNWU81732J',
          fromMe: true,
          timestamp: '1654823909'
        }
      ]
    }
  },
  jid
)
```

Delete a Chat

```
const lastMsgInChat = await getLastMessageInChat(jid) // implement 1
await sock.chatModify({
  delete: true,
  lastMessages: [
    {
      key: lastMsgInChat.key,
      messageTimestamp: lastMsgInChat.messageTimestamp
    }
  ]
},
jid
)
```

Pin/Unpin a Chat


```
await sock.chatModify({
  pin: true // or `false` to unpin
},
jid
)
```

Star/Unstar a Message

```
await sock.chatModify({
  star: {
    messages: [
      {
        id: 'messageID',
        fromMe: true // or `false`
      }
    ],
    star: true // - true: Star Message; false: Unstar Message
  }
},
jid
)
```

Disappearing Messages

- Ephemeral can be:

Time	Seconds
Remove	0
24h	86.400
7d	604.800
90d	7.776.000

- You need to pass in **Seconds**, default is 7 days

```
// turn on disappearing messages
await sock.sendMessage(
  jid,
  // this is 1 week in seconds -- how long you want messages to disappear
  { disappearingMessagesInChat: WA_DEFAULT_EPHEMERAL }
)

// will send as a disappearing message
await sock.sendMessage(jid, { text: 'hello' }, { ephemeralExpiration: 604800 })

// turn off disappearing messages
await sock.sendMessage(
  jid,
  { disappearingMessagesInChat: false }
)
```

User Querys

Check If ID Exists in Whatsapp

```
const [result] = await sock.onWhatsApp(jid)
if (result.exists) console.log(`${jid} exists on WhatsApp, as jid: ${jid}`)
```

Query Chat History (groups too)

- You need to have oldest message in chat

```
const msg = await getOldestMessageInChat(jid)
await sock.fetchMessageHistory(
  50, //quantity (max: 50 per query)
  msg.key,
  msg.messageTimestamp
)
```

- Messages will be received in `messaging-history.set` event

Fetch Status

```
const status = await sock.fetchStatus(jid)
console.log('status: ' + status)
```

Fetch Profile Picture (groups too)

- To get the display picture of some person/group

```
// for low res picture
const ppUrl = await sock.profilePictureUrl(jid)
console.log(ppUrl)

// for high res picture
const ppUrl = await sock.profilePictureUrl(jid, 'image')
```

Fetch Bussines Profile (such as description or category)

```
const profile = await sock.getBusinessProfile(jid)
console.log('business description: ' + profile.description + ', cate
```



Fetch Someone's Presence (if they're typing or online)

```
// the presence update is fetched and called here
sock.ev.on('presence.update', console.log)

// request updates for a chat
await sock.presenceSubscribe(jid)
```

Change Profile

Change Profile Status

```
await sock.updateProfileStatus('Hello World!')
```

Change Profile Name

```
await sock.updateProfileName('My name')
```

Change Display Picture (groups too)

- To change your display picture or a group's

[!NOTE] Like media messages, you can pass { stream: Stream } or { url: Url } or Buffer directly, you can see more [here](#)

```
await sock.updateProfilePicture(jid, { url: './new-profile-picture.'
```

Remove display picture (groups too)

```
await sock.removeProfilePicture(jid)
```

Groups

- To change group properties you need to be admin

Create a Group

```
// title & participants
const group = await sock.groupCreate('My Fab Group', ['1234@s.whats:
console.log('created group with id: ' + group.gid)
await sock.sendMessage(group.id, { text: 'hello there' }) // say he
```

Add/Remove or Demote/Promote

```
// id & people to add to the group (will throw error if it fails)
await sock.groupParticipantsUpdate(
  jid,
  ['abcd@s.whatsapp.net', 'efgh@s.whatsapp.net'],
  'add' // replace this parameter with 'remove' or 'demote' or 'pr
)
```

Change Subject (name)


```
await sock.groupUpdateSubject(jid, 'New Subject!')
```

Change Description

```
await sock.groupUpdateDescription(jid, 'New Description!')
```

Change Settings

```
// only allow admins to send messages
await sock.groupSettingUpdate(jid, 'announcement')
// allow everyone to send messages
await sock.groupSettingUpdate(jid, 'not_announcement')
// allow everyone to modify the group's settings -- like display pic
await sock.groupSettingUpdate(jid, 'unlocked')
// only allow admins to modify the group's settings
await sock.groupSettingUpdate(jid, 'locked')
```



Leave a Group

```
// will throw error if it fails
await sock.groupLeave(jid)
```

Get Invite Code

- To create link with code use 'https://chat.whatsapp.com/' + code

```
const code = await sock.groupInviteCode(jid)
console.log('group code: ' + code)
```

Revoke Invite Code

```
const code = await sock.groupRevokeInvite(jid)
console.log('New group code: ' + code)
```

Join Using Invitation Code

- Code can't have `https://chat.whatsapp.com/` , only code

```
const response = await sock.groupAcceptInvite(code)
console.log('joined to: ' + response)
```

Get Group Info by Invite Code

```
const response = await sock.groupGetInviteInfo(code)
console.log('group information: ' + response)
```

Query Metadata (participants, name, description...)

```
const metadata = await sock.groupMetadata(jid)
console.log(metadata.id + ', title: ' + metadata.subject + ', descri:
```



Join using groupInviteMessage

```
const response = await sock.groupAcceptInviteV4(jid, groupInviteMes:
console.log('joined to: ' + response)
```



Get Request Join List

```
const response = await sock.groupRequestParticipantsList(jid)
console.log(response)
```

Approve/Reject Request Join

```
const response = await sock.groupRequestParticipantsUpdate(
  jid, // group id
  ['abcd@s.whatsapp.net', 'efgh@s.whatsapp.net'],
  'approve' // or 'reject'
)
console.log(response)
```

Get All Participating Groups Metadata

```
const response = await sock.groupFetchAllParticipating()  
console.log(response)
```

Toggle Ephemeral

- Ephemeral can be:

Time	Seconds
Remove	0
24h	86.400
7d	604.800
90d	7.776.000

```
await sock.groupToggleEphemeral(jid, 86400)
```

Change Add Mode

```
await sock.groupMemberAddMode(  
  jid,  
  'all_member_add' // or 'admin_add'  
)
```

Privacy

Block/Unblock User

```
await sock.updateBlockStatus(jid, 'block') // Block user  
await sock.updateBlockStatus(jid, 'unblock') // Unblock user
```

Get Privacy Settings

```
const privacySettings = await sock.fetchPrivacySettings(true)  
console.log('privacy settings: ' + privacySettings)
```

Get BlockList

```
const response = await sock.fetchBlocklist()  
console.log(response)
```

Update LastSeen Privacy

```
const value = 'all' // 'contacts' | 'contact_blacklist' | 'none'  
await sock.updateLastSeenPrivacy(value)
```

Update Online Privacy

```
const value = 'all' // 'match_last_seen'  
await sock.updateOnlinePrivacy(value)
```

Update Profile Picture Privacy

```
const value = 'all' // 'contacts' | 'contact_blacklist' | 'none'  
await sock.updateProfilePicturePrivacy(value)
```

Update Status Privacy

```
const value = 'all' // 'contacts' | 'contact_blacklist' | 'none'  
await sock.updateStatusPrivacy(value)
```

Update Read Receipts Privacy

```
const value = 'all' // 'none'  
await sock.updateReadReceiptsPrivacy(value)
```

Update Groups Add Privacy

```
const value = 'all' // 'contacts' | 'contact_blacklist'  
await sock.updateGroupsAddPrivacy(value)
```

Update Default Disappearing Mode

- Like **this**, ephemeral can be:

Time	Seconds
Remove	0
24h	86.400
7d	604.800
90d	7.776.000

```
const ephemeral = 86400
```

```
await sock.updateDefaultDisappearingMode(ephemeral)
```

Broadcast Lists & Stories

Send Broadcast & Stories

- Messages can be sent to broadcasts & stories. You need to add the following message options in sendMessage, like this:

```
await sock.sendMessage(  
  jid,  
  {  
    image: {  
      url: url  
    },  
    caption: caption  
  },  
  {  
    backgroundColor: backgroundColor,  
    font: font,  
    statusJidList: statusJidList,  
    broadcast: true  
  }  
)
```

- Message body can be a `extendedTextMessage` or `imageMessage` or `videoMessage` or `voiceMessage`, see [here](#)
- You can add `backgroundColor` and other options in the message options, see [here](#)
- `broadcast: true` enables broadcast mode
- `statusJidList`: a list of people that you can get which you need to provide, which are the people who will get this status message.
- You can send messages to broadcast lists the same way you send messages to groups & individual chats.
- Right now, WA Web does not support creating broadcast lists, but you can still delete them.
- Broadcast IDs are in the format `12345678@broadcast`

Query a Broadcast List's Recipients & Name

```
const bList = await sock.getBroadcastListInfo('1234@broadcast')
console.log(`list name: ${bList.name}, recps: ${bList.recipients}`)
```

Writing Custom Functionality

Baileys is written with custom functionality in mind. Instead of forking the project & re-writing the internals, you can simply write your own extensions.

Enabling Debug Level in Baileys Logs

First, enable the logging of unhandled messages from WhatsApp by setting:

```
const sock = makeWASocket({
  logger: P({ level: 'debug' }),
})
```

This will enable you to see all sorts of messages WhatsApp sends in the console.

How Whatsapp Communicate With Us

[!TIP] If you want to learn whatsapp protocol, we recommend to study about Libsignal Protocol and Noise Protocol

- **Example:** Functionality to track the battery percentage of your phone. You enable logging and you'll see a message about your battery pop up in the console:

```
{
  "level": 10,
  "fromMe": false,
  "frame": {
    "tag": "ib",
    "attrs": {
      "from": "@s.whatsapp.net"
    },
    "content": [
      {
        "tag": "edge_routing",
        "attrs": {},
        "content": [
          {
            "tag": "routing_info",
            "attrs": {},
            "content": {
              "type": "Buffer",
              "data": [8,2,8,5]
            }
          }
        ]
      }
    ]
  },
  "msg": "communication"
}
```

The 'frame' is what the message received is, it has three components:

- tag -- what this frame is about (eg. message will have 'message')
- attrs -- a string key-value pair with some metadata (contains ID of the message usually)
- content -- the actual data (eg. a message node will have the actual message content in it)
- read more about this format [here](https://www.npmjs.com/package/baileys#non-media-messages)

Register a Callback for Websocket Events

[!TIP] Recommended to see `onMessageReceived` function in `socket.ts` file to understand how websockets events are fired

```
// for any message with tag 'edge_routing'  
sock.ws.on('CB:edge_routing', (node: BinaryNode) => { })  
  
// for any message with tag 'edge_routing' and id attribute = abcd  
sock.ws.on('CB:edge_routing,id:abcd', (node: BinaryNode) => { })  
  
// for any message with tag 'edge_routing', id attribute = abcd & fi  
sock.ws.on('CB:edge_routing,id:abcd,routing_info', (node: BinaryNode)
```

[!NOTE] Also, this repo is now licenced under GPL 3 since it uses **libsignal-node**

Keywords

whatsapp js-whatsapp whatsapp-api whatsapp-web whatsapp-chat
whatsapp-group automation multi-device

Install

```
> npm i baileys
```



Repository

github.com/WhiskeySockets/Baileys

Homepage

github.com/WhiskeySockets/Baileys

Weekly Downloads

7.710



Version
6.7.16

License
MIT

Unpacked Size
8.24 MB

Total Files
171

Last publish
a month ago

Collaborators



>Try on RunKit

Report malware



- Support
- Help
- Advisories

Status

Contact npm

Company

About

Blog

Press

Terms & Policies

Policies

Terms of Use

Code of Conduct

Privacy