aws

Bots and Integrations Guide

AWS Wickr

AWS Wickr: Bots and Integrations Guide

Table of Contents

What are AWS Wickr bots?

AWS Wickr bots are powerful tools that enable external services and workflows to communicate seamlessly with AWS Wickr users. These bots function as standard users within the AWS Wickr ecosystem, allowing administrators to apply consistent security controls while enabling end-users to interact through messaging. Bots can send and receive messages, handle file attachments, manage rooms and groups, and add interactive UI elements for enhanced user engagement.

Implemented using either a Node.js native library or REST API, AWS Wickr bots offer versatile integration capabilities. They can manage webhooks, interact with file repositories, leverage AWS services like Rekognition for image analysis, bridge communications with platforms such as Slack, Discord, or any Matrix-compatible endpoint, integrate with AI and LLMs, broadcast messages, and even gather location data for mapping purposes. The Matrix integration capability is particularly valuable for organizations requiring federated communication across different platforms while maintaining security protocols.

Use cases for AWS Wickr bots are diverse, ranging from mass communication and emergency notifications to AI-powered chatbots and cross-platform messaging solutions. For instance, enterprises can use bots to automate compliance processes, manage file systems, or create custom integrations with their existing tools. A prime example of AWS Wickr bots in action is showcased in the "Operation Recovery" case study, where bots played a crucial role in coordinating life-saving efforts during critical scenarios.

By leveraging AWS Wickr bots, organizations can extend the platform's capabilities, streamline workflows, and create tailored solutions that meet their specific communication and integration needs, all while maintaining the robust security framework inherent to AWS Wickr. The following sections provide detailed information about getting started with bots and the available integrations. Please note that setting up and configuring bots requires developer involvement and familiarity with command-line operations.

AWS Wickr bot capabilities

AWS Wickr bots can do the following:

- Send and receive messages
- Send and receive file attachments
- Create and manage AWS Wickr rooms and groups

- Add UI elements for user interaction

> **Important**
>
> AWS Wickr bots cannot initiate or join calls at this time.

AWS Wickr bots utilize either Node.js code via a native library or a REST API to control what the bot can do. Examples of what bots can do are:

- Webhooks
- Manage a file repository
- Send images or video to AWS Recognition for analysis
- Send and receive messages on another platform, like Slack, Discord, or a Matrix compatible endpoint
- Interact with Generative AI and LLMs
- Broadcast messages or file to any number of AWS Wickr users within your network
- Gather user locations to build a map view

Setting up for Wickr IO

This section is intended for systems administrators and/or developers to deploy a Wickr-provided bot within their Wickr network or to build and integrate a custom bot of their own.

The following sections will be of interest to all users:

- *Quick start*: Describes a quick guide from creation to running and interacting with your Wickr bot clients.
- Available Bot Integrations: Contains descriptions of current Wickr IO integrations that you can use.
- Wickr IO Command Line Interface (CLI): Describes all of the commands that are available from the Wickr IO command line interface.

Developers have access to several levels of APIs when developing Wickr IO integration software. Familiarity with all aspects of the Wickr IO system will be helpful when developing Wickr IO integrations. The following sections give details on the available APIs and any other integration components you will need to deal with:

- Wickr IO Node.js Addon API: The main software API you will use to access the Wickr messaging capabilities.
- Wickr IO bot API: A higher level software API that provides additional capabilities above the main addon APIs, basically a bot development toolkit.
- Integration setup: Information about integration software modules you will need to maintain that will be used to integrate your Wickr IO integration into the Wickr IO Integration Gateway.
- Wickr IO Web Interface Integration: Information about how to use the REST API as a way to integrate your software.

If you are planning on developing Wickr IO integrations, you should read this entire document. Administrators may skip the development sections.

Prerequisites

Before you start, complete the following before continuing with this guide:

- Sign up for AWS Wickr. For more information, see Setting up for AWS Wickr in the *AWS Wickr Administration Guide*..
- Create and configure your AWS Wickr network. For more information, see Create a network in the *AWS Wickr Administration Guide*.
- Provision a host machine to run the Wickr IO bot Docker container.

Host machine and requirements

You should provision a host machine capable of running Docker to deploy the Wickr IO Docker container. Refer to the Docker website for instructions on how to install Docker on your system. Once the docker is installed, make sure the docker service is up and running using the `docker info` command.

While exact requirements may vary based on your deployment scenario, the following recommendations should be considered as baseline specifications for provisioning the host system:

Host OS specifications

The Wickr IO Docker container can be deployed on any Docker capable machine (like Amazon EC2 instances, VMs, etc.). However, since most AWS testing an validation is performed on Ubuntu 20.04 and Amazon Linux, we recommend using either of these operating systems for compatibility and stability.

The Wickr IO Docker image uses the Ubuntu Linux operating system.

Host resource specifications

At a minimum the Wickr IO container should have the following resources available when deploying and running a single bot:

- 4 GB RAM
- 2 CPU
- 8GB+ disk space

> **Note**
>
> Increase these resources to ensure availability when running multiple bots or more intensive local workloads. Make sure to regularly monitor your host's disk space and memory utilization to avoid disruptions in container performance.

Networking requirements

Wickr IO bot clients have the same networking requirements as traditional Wickr clients. The ports and domains needed for basic connectivity are in the AWS Wickr administration guide. For more information, see Ports and domains to allow list for your Wickr network.

Depending on your use case you may have to start the Docker container in a way to allow access to specific network ports or network proxy settings as well. For example, the Wickr Web Interface integration requires a TCP port to expose the REST API.

If you plan to use other AWS services with a Wickr bot, you must ensure the host has the appropriate AWS Identity and Access Management (IAM) role and policy to access them. For more information, see What is IAM?.

Persistent Data

The Wickr IO Docker container will also require access to the host file system in order to save persistent data. This is necessary to stop or upgrade the image without losing the state of your Wickr IO clients. You will need to specify this location to the Docker image when you run it.

Security Recommendations

We recommend following best practices and if applicable, your organization's security policies to secure your bot deployment. This can include, but isn't restricted to, firewall rules, host system access auditing, regular host system OS updates, and monitoring. We designed Wickr IO bots to be both powerful and flexible for custom use cases and while they inherit many security protections from our standard Wickr clients, it falls on you to secure the host system appropriately to protect it (and your bot) from unauthorized access. For more information on shared responsibility, see Shared Responsibility Model.

The Wickr IO bot container is moderately hardened to remove unnecessary services, etc., but its threat model assumes that it is deployed on an internal network segment and configured as a

client system. Don't forget to augment your security controls as you expand your use cases. The bottom line is if you lose control of your bot host, you will likely lose control of your bot and all the data. Log only what you need to log, encrypt what you need to encrypt, and use strong access controls.

Bots are not isolated by default. Users outside your network can interact with bots if they guess the bot username and your security group allows external federation. For more information, see Security groups for AWS Wickr .

Installation

Once you have a host machine setup and Docker installed, the installation of Wickr IO software is as simple as pulling the appropriate Wickr IO Docker container. For more information on using Docker, see Get Started with Docker.

The Wickr IO Docker images are hosted on Public ECR. You can pull down the Wickr IO Docker image located at the following public ECR repository:

Wickr IO bot cloud.

The docker image can be pulled down to the host machine using the following command:

```
docker pull public.ecr.aws/x3s2s6k3/wickrio/bot-cloud:latest
```

Note

Depending on your use case, you may want to pull a particular version. For more information on available versions, see Wickr IO bot cloud.

In case, you do not have access to public ECR, the docker image can also be pulled down, from the Dockerhub repository, to the host machine using the following command:

```
docker pull wickr/bot-cloud:latest
```

> **Note**
>
> The Docker Hub repository will soon be deprecated and the bot images will only be available on public ECR.

Wickr IO Components

The Wickr IO Docker container contains the Wickr IO client software and the client service and configuration software. The Wickr IO client, as opposed to the Wickr client, provides a software interface (API) to the Wickr capabilities, instead of through a graphical user interface (GUI). This software interface can be leveraged either through the native Node.js add-on APIs or an optional REST API (using the Web Interface integration) to provide the ability to send and receive messages, as well as create secure rooms and group conversations.

The Wickr IO bot software is distributed as a Docker image. There are different Docker images depending on the type of Wickr environment you are using. Each Wickr IO Docker container has the following components:

- **Wickr IO client**: The Wickr IO client is a Wickr client that interacts with integration software through an API. Using this software API, the integration software can access the Wickr messaging capabilities. The Wickr IO client is associated with a Wickr user account and interacts with other Wickr clients as a Wickr bot user.
- **Wickr IO Node.js API**: This software provides an API that allows native Node.js programs (integrations) to interact with the Wickr IO client. This software is distributed via the public Node Package Manager (NPM) registry. For more information, see Node.js Addon API.
- **Wickr IO Integration Software**: The Wickr IO integration software is written using Node.js and uses the Wickr IO Node-js add-on API to implement the integration functionality. The Wickr IO Bot Docker container includes a few Wickr IO integrations that are maintained by AWS Wickr team, and a few sample integrations are available on the public NPM Registry. You can use the sample integrations as examples to build your own. For more information, see Available Bot Integrations.
- **Wickr IO console**: This console software provides a command line interface that is used to maintain the operation of your Wickr IO clients. The command line interface supports adding, modifying, deleting Wickr IO clients and the associated integration software. You will also use the command line interface to start and stop running the Wickr IO clients. For more information, see CLI commands.

Version 6.48 announcement

Version 6.48 of the Bots docker image contains the upgrade to Node 20. If you are using any bots, you will need to make the following modifications to them to ensure they work with this latest bot version. Upgrading to the new version without completing these steps will disrupt the functionality of your custom bots:

1. If you are creating custom bots/integrations, you will need to update the integration to include the following changes:
   - Bump wickrio-bot-api version to 7.1.x. (If you have wickrio_addon as a dependency, it should also be bumped to 7.1.x).
   - Remove occurrences of Node 16 usage.
   - Make changes to work with asynchronous APIs.

   This is an example of changes to be made to any custom integration to accommodate compatibility with version 6.48 (and later).
2. If you are using any of the officially supported integrations, please make sure to upgrade your integrations to the latest version using `upgrade` command in Docker CLI. For more information, see the section called “Upgrading bots”.

As of version 6.48, the Bots docker image has the following major changes:

1. Upgraded to use Node 20 (previously used Node 16).
2. Deprecated multiple integrations. This is the list of officially supported integrations:
   - wickrio-broadcast-bot
   - wickrio_web_interface_bot
   - wickrio-compliance-bot (only available for Enterprise environments)
3. This is the list of sample integrations that can be pulled from NPM registry for testing purposes:
   a. @wickr-sample-integrations/wickrio-hello-world-bot
   b. @wickr-sample-integrations/wickrio-example-app
   c. @wickr-sample-integrations/wickrio-lex-bot
   d. @wickr-sample-integrations/wickrio-rekognition-bot
   e. @wickr-sample-integrations/wickrio-translation-bot

4. The WickrIO addon has been updated to use ZeroMq to interact with WickrIO client, making the WickrIO APIs asynchronous.

Quick start

This guide walks you through deploying your first bot in Wickr. When you complete this guide, you'll have a bot in Wickr that can respond to pre-configured messages.

Topics

Prerequisites

Before you start, be sure to complete the following prerequisites if you haven't already:

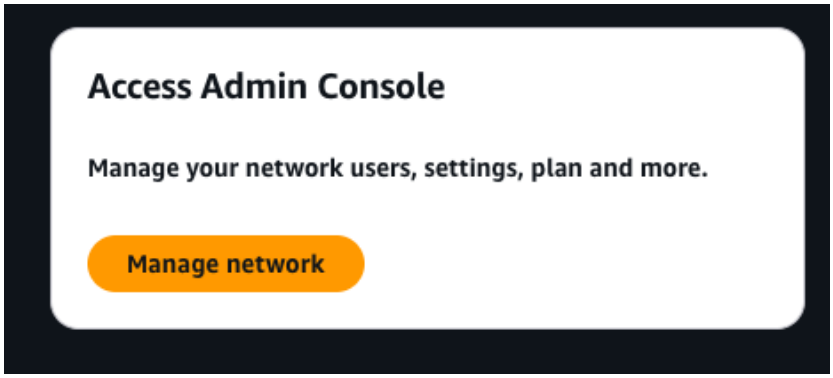
- A host that meets the requirements specified in Host machine and requirements.
- An AWS Console account with access to Wickr (Commercial and Gov): Getting started with AWS Wickr
- Create a Wickr network:
  - Getting started (Commercial and Gov)
  - Network provisioning (Enterprise)

Step 1: Create a bot user

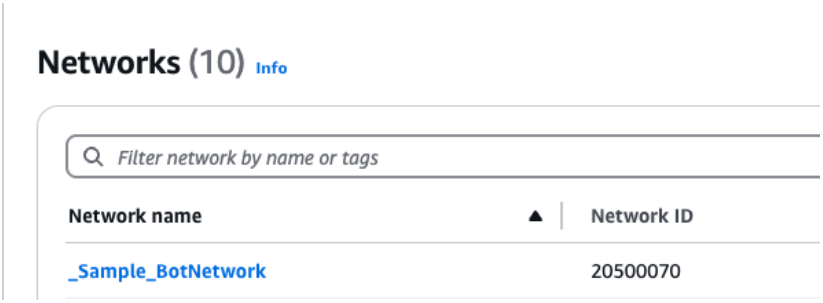
You can create a bot user in the Wickr console.

Complete the following procedure to create a bot user.

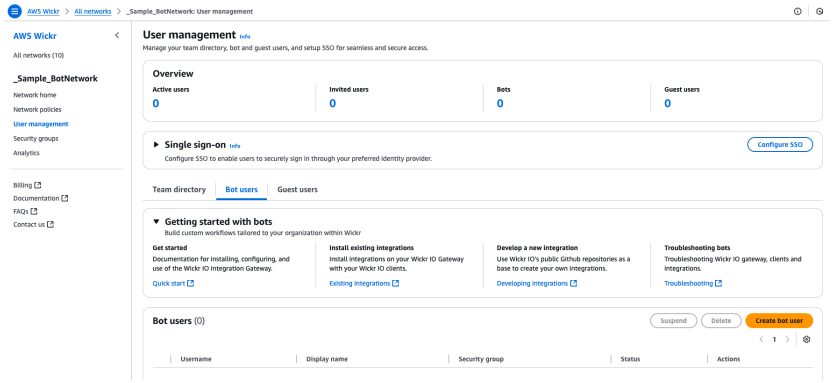
1. Log in to the AWS Wickr console at https://console.aws.amazon.com/wickr/
2. In the **Access Admin Console** section of the page, choose **Manage network**.



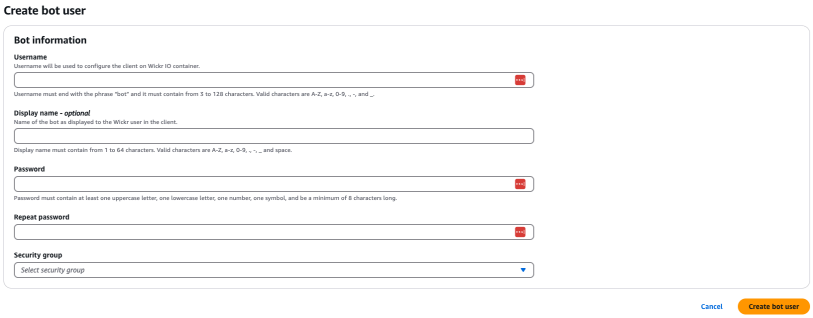
3. Select your Wickr network by finding it using its Network name or Network ID on the Manager network page. If necessary, search for the network by its Network name.



4. In the left navigation, choose **User management** to access the User Management page. This page allows you to add, remove, and set properties for all kinds of Wickr user types including licensed users, bots, and guest users.



5. Choose the **Bot users** tab and choose **Create bot user**.



6. Enter the following information:

   - **Username** — The internal username of the bot.
   - **Display name** — The bot name that will be shown to end users.
   - **Password** — The password that you'll need to register and log in to the bot.
   - **Security group** — Controls permissions for the bot and how it's allowed to communicate.

Step 2: Configure the host

Complete the following procedure to configure the host.

1. Connect to your EC2 or host machine and create a directory for your Docker volume. For this example, we'll use a directory in the current user's home, `~/WickrIO`. This directory will mount inside the container at `/opt/WickrIO`.

   ```
   cd ~
   mkdir WickrIO
   ```

2. Make sure that the `~/WickrIO` directory or your chosen directory is shared with Docker. For more information, see Sharing local files with containers.

3. Pull the public Wickr IO Docker image:

   ```
   docker pull public.ecr.aws/x3s2s6k3/wickrio/bot-cloud:latest
   ```

   Note

   If you have a Docker image saved in in .tar format, you must load it using the following command before starting the docker container:
   `docker load -i image-name.tar`

Replace *image-name* with the actual file name of your Docker image.

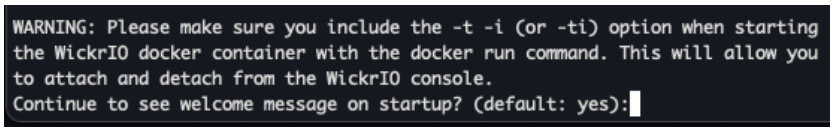
Step 3: Deploy and configure the Docker container

Complete the following procedure to deploy and configure the Docker container.

1. Start the Docker image on your host:

```
docker run -v ~/WickrIO:/opt/WickrIO -ti public.ecr.aws/x3s2s6k3/wickrio/bot-cloud:latest
```

2. Select your preference for the welcome message.

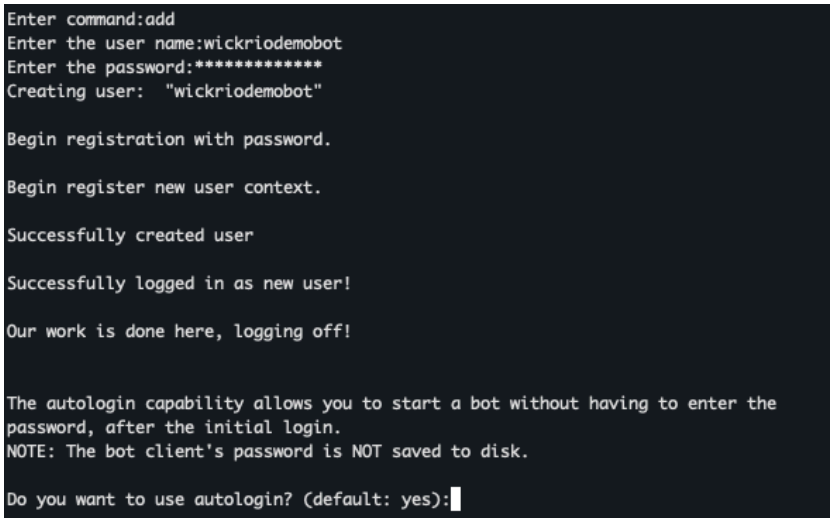


Deploy an existing bot

Complete the following procedure to deploy an existing bot.

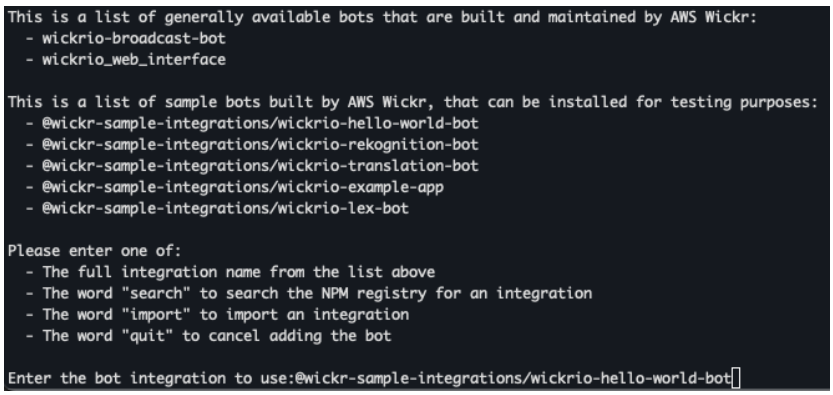
Wickr IO Hello World Bot

1. Follow Steps 1-3 in *Quick start* to get the bot container created and running.
2. At the **Enter command:** prompt, enter the command **add**.



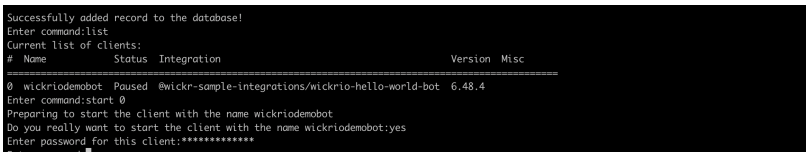
3. Over the next several prompts, enter the **username** and **password** created in the previously.

4. Next you are prompted to select an integration. In the sample, we used `@wickr-sample-integrations/wickrio-hello-world-bot`.

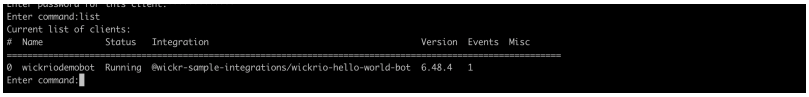


5. Start the bot by doing the following:

   1. Use the **list** command to view a list of available bots.

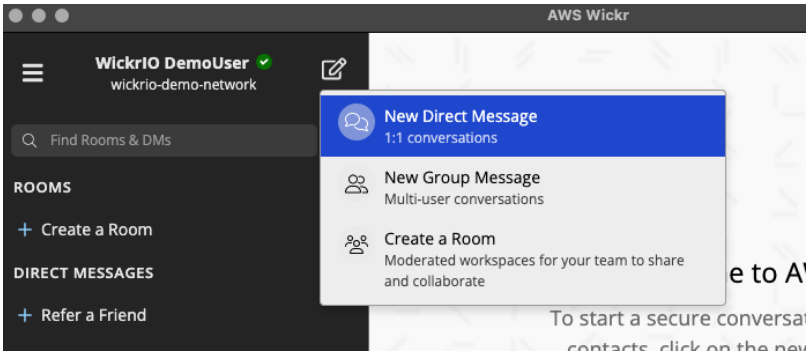


   2. Using the number of the bot that you just created (0 in the example), type the command **start #**, where # is the bot number (0 in the example).
   3. Enter the password for the bot.
   4. Wait several seconds, and then use the **list** command again to verify that the bot is running.

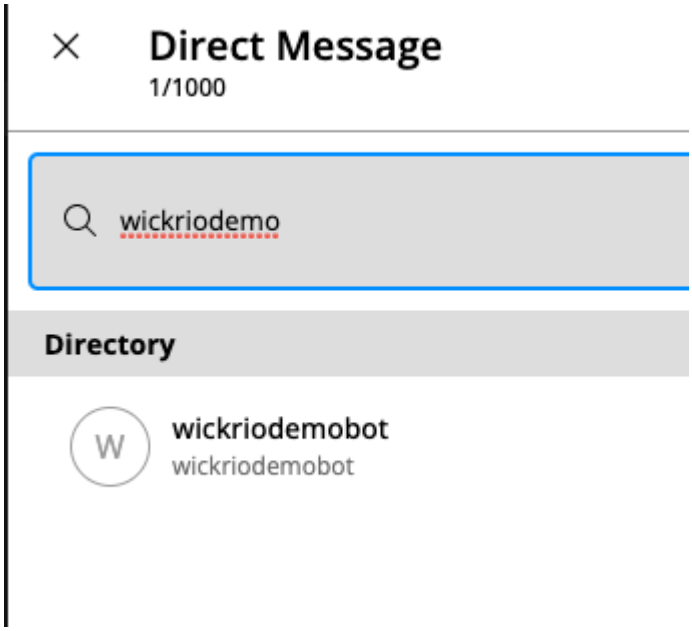


6. Interact with the bot:

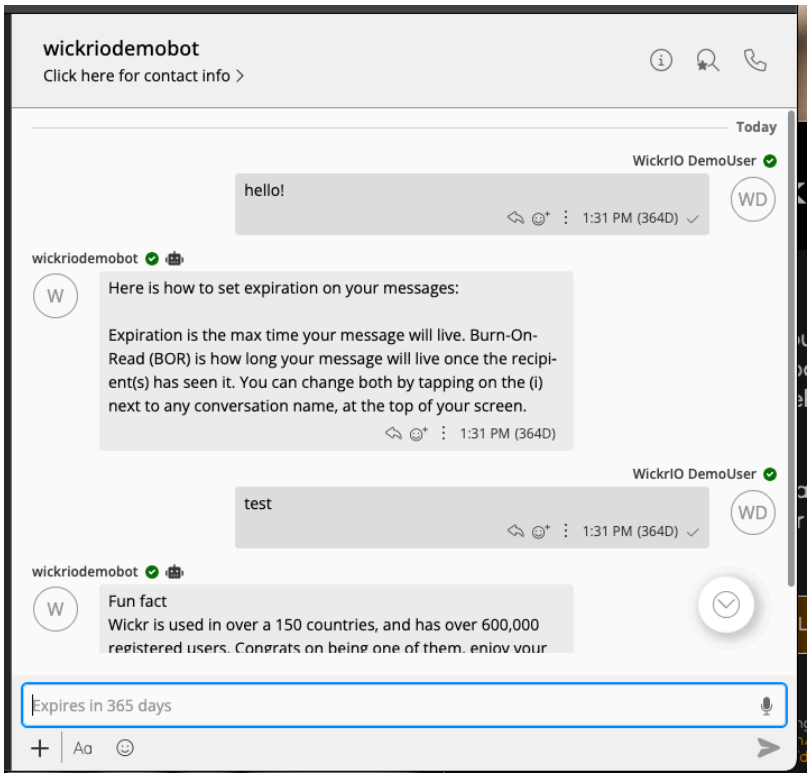
   1. Using your Wickr user, choose the **New Direct Message** button.



   2. In the search bar, search for your bot by display name.



3. Select your bot for a direct message, and send a message.



AWS Wickr managed Integrations

This section describes the production-ready bots that are built, maintained, and dispersed by AWS Wickr. The Wickr team QAs them at a regular cadence to ensure optimal performance and compatibility with Wickr apps.

The following is a list of available bots which are included in the Wickr IO bot image:

- **Broadcast bot** - This integration allows a user or administrator to quickly send or broadcast messages to any number of users. These messages can prompt recipients for acknowledgement, prompt them to share their location with the sender, and repeat until the recipient confirms receipt. For more information, see BroadcastBot Integration.
- **Web-Interface bot** - This integration is the official Wickr IO Web API tool. Provides an endpoint to send HTTP requests to communicate with the Wickr IO API. This also makes it possible to create Wickr integrations using any programming language. For more information, see Web Interface Integration

BroadcastBot Integration

The Wickr IO BroadcastBot allows you to broadcast messages to all of the members of your network or specific security groups. The messages will be sent to each individual within the network or security group using Wickr 1on1 conversations. You can broadcast either messages, files or voice memos. Users that receive a broadcast from the BroadcastBot can also acknowledge receiving messages (using the /ack command).

The BroadcastBot will maintain the status for each message sent on a per user basis. You can retrieve a detailed or summary report of the status for the broadcasts that you have sent.

You can interact with the BroadcastBot using any one of the following types of applications:

- Wickr interface - Interact with the BroadcastBot using Wickr commands.
- Web interface - Interact with the BroadcastBot using a web-based interface.
- REST API interface - Interact with the BroadcastBot using REST APIs.

The Wickr interface supports a set of commands that allow you to send broadcast messages and retrieve status information associated with the broadcast messages you send. You will interact with

the BroadcastBot on a Wickr 1on1 conversation. Details of the supported commands are described in the section called “BroadcastBot Wickr Interface”.

The web interface is a web-based application that allows you to interact with the BroadcastBot. The web interface is initiated by a command you enter in the BroadcastBot's Wickr 1on1 conversation. Details of the commands and interaction with the BroadcastBot via the web interface are described in the section called “BroadcastBot Web Interface”.

The REST API interface is provided to allow you to easily integrate the BroadcastBot features into your own applications. The REST API provides the same commands to interact with the BroadcastBot as the other methods of interacting with the BroadcastBot. Details of the REST APIs are described in the section called “BroadcastBot REST API”.

The Wickr IO BroadcastBot is a public integration. See the section called “Broadcast Bot Installation” or the section called “Broadcast Bot Enterprise Installation” for details on installation and configuration of the specific Wickr Docker images.

Broadcast Bot Installation

This section describes the requirements and configuration of the BroadcastBot for Wickr networks.

Requirements

Before you can install and configure the BroadcastBot, you will need to create a Wickr Bot user via the appropriate Wickr admin console. The the section called “Step 1: Create a bot user” section describes how to create a Wickr IO client on the Wickr Admin console.

The following is a list of tokens that are required to start/configure a Wickr BroadcastBot integration:

- `WICKRIO_BOT_NAME` - The name of the BroadcastBot Wickr client. This value should be automatically set.
- `DATABASE_ENCRYPTION_CHOICE` - Identifies if you want to encrypt the BroadcastBot information. Choices are 'yes' or 'no'.
- `DATABASE_ENCRYPTION_KEY` - A random sequence of bytes used to encrypt the BroadcastBot information. This string must be at least 16 characters in length. Entering a value with less than 16 characters will not encrypt the BroadcastBot information. It is not required to be encrypted but is highly recommended.

- ADMINISTRATORS - A comma separated list of Wickr users that can use the BroadcastBot to send broadcast messages. These are the only users that can use the BroadcastBot. Additional admin users can be added via the /admin command.
- WEB_INTERFACE - Identifies if you want to use the Web Interface and/or the REST API interface. Choices are 'yes' or 'no'.
- WEB_APPLICATION - Identifies if you want to use the Web Interface. Choices are 'yes' or 'no'. This is only valid if the WEB_INTERFACE is 'yes'.
- WEBAPP_HOST - This is the public IP address or host that users will use to get access to the Web Interface application.
- WEBAPP_PORT - This is the public IP port that users will use to get access to the Web Interface application.
- REST_APPLICATION - Identifies if you want to use the REST API. Choices are 'yes' or 'no'. This is only valid if the WEB_INTERFACE is 'yes'.
- BOT_PORT - This is the docker internal port that is used to interface with the web applications.
- BOT_KEY - The API Key that is used in every endpoint call. This is the <api key=""> value that is contained in every endpoint URL, as is shown in the table in the previous section.
- BOT_AUTH_TOKEN - The authentication string used to generate the Base64 value to be sent in the authorization field of the HTTP Header (Recommended: 24-character alphanumeric string). You will need to generate a Base64 value of this token and the add it to the HTTP authorization header (i.e. Basic MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNA==).
- HTTPS_CHOICE - Identifies if you will be using HTTPS to interact with the web applications of this bot. Choices are 'yes' or 'no'.
- SSL_KEY_LOCATION - Full path name of the .key file (only required if the HTTPS_CHOICE is 'yes'). The file must be located in the shared directory that the integration software running on the Docker image can access.
- SSL_CRT_LOCATION - Full path name of the .cert file (only required if the HTTPS_CHOICE is 'yes'). The file must be located in the shared directory that the integration software running on the Docker image can access.
- BROADCAST_ENABLED - Enter 'yes' to enable the /broadcast command and allow broadcasts to the whole network and security groups. Enter 'no' to only allow sending to files with the /send command.

Configuration

You will need to have some familiarity with Docker in order to configure and start the Wickr BroadcastBot. The the section called “Step 3: Deploy and configure the Docker container” section has some helpful information on working with Wickr Docker images.

The following are steps you can use to create a BroadcastBot integration:

1. Start the docker image, which will download the docker image, if needed. Note: assign a unique name to this docker image which you can use for other docker commands, also note there are 2 ports opened (4002 and 8080) with the following docker run command:

   ```
   docker run -v /opt/WickrIOShare:/opt/WickrIO -p 4002:4002 -p 8080:8080 \
   --d --restart=always --name="MyBCastBot" -ti \
   public.ecr.aws/x3s2s6k3/wickrio/bot-cloud:latest
   ```

   > **Note**
   >
   > If you are on Wickr GovCloud, please remember to modify the above command to replace docker image link with GovCloud image i.e `public.ecr.aws/x3s2s6k3/wickrio/bot-cloud-govcloud:latest`

2. Attach to the docker image, using the name from the previous step: `docker attach MyBCastBot`
3. Agree to the license agreement, if you have not already done so.
4. Enter the add command at the prompt, filling in the username and password you created in step one.
5. Enable autologin.
6. Enter the broadcast bot integration from the list, for example `wickrio-broadcast-bot`
7. Respond to the configuration prompts with appropriate values, see the sample output below.
8. The broadcast bot is configured now. Start the client by entering `start` and then `y` and then the password.

If you have followed along so far you now have the broadcast bot running on your network!

The following is sample output from adding a broadcast bot:

```
Enter command:add
Enter the user name:bcast_bot
Enter the password:********
Creating user:  "bcast_bot"

Begin registration with password.
Begin register new user context.

Begin register existing user context.

Successfully created user

Successfully logged in as new user!

Our work is done here, logging off!

Return code from provision is: 0

The autologin capability allows you to start a bot without having to enter the
password, after the initial login.
NOTE: The bot client's password is NOT saved to disk.

Do you want to use autologin? (default: yes):
Searching NPM registry
Searching NPM registry
Searching NPM registry

These integrations are local:
  - hubot

These integrations are from the NPM registry:
  - wickrio_web_interface
  - wickrio-file-bot
  - wickrio-hello-world-bot
  - wickrio-example-app
  - wickrio-broadcast-bot

Please enter one of:
  - The full integration name from the list above
  - The word "search" to search the NPM registry for an integration
  - The word "import" to import an integration
  - The word "quit" to cancel adding the bot
```

```
Enter the bot integration to use:wickrio-broadcast-bot
*****************************************************************
Begin setup of wickrio-broadcast-bot software for bcast_bot
Copying wickrio-broadcast-bot software

Installing wickrio-broadcast-bot software
Installing
Installing
Begin configuration of wickrio-broadcast-bot software for bcast_bot
Adding ADMINISTRATORS to the list of tokens
Adding VERIFY_USERS to the list of tokens
 Do you want to encrypt the configuration values [yes|no]: (no) :
 Do you want to setup the web interface (REST API or WEB Application) [yes/no]:
 (no) :yes
 Do you want to use the web application [yes/no]: (no) :yes
 Please enter the host name or ip address to reach the web application:
 (false) :54.1.2.3
 Please enter the host port to use to reach the web application: (false) :8080
 Do you want to use the REST application [yes/no]: (no) :yes
 Please enter your client bot's port: (false) :4002
 Please enter your client bot's API-Key: (false) :ABCDEF
 Please create an Web API Basic Authorization Token, we recommend an alphanumeric
 string with at least 24 characters: (false) :12345678901234567890ABCD
 Do you want to set up an HTTPS connection with the Web API Interface, highly
 recommended [yes/no]: (no) :
 Do you want to map users locations when you send broadcasts [yes/no]: (no) :
 Enter the list of administrators: (N/A) :auser01,auser02
 Enter the mode to verify users: (automatic) :
Finished Configuring!

Integration files written to:
/opt/WickrIO/clients/bcast_bot/integration/wickrio-broadcast-bot

End of setup of wickrio-broadcast-bot software for bcast_bot
*****************************************************************
Successfully added record to the database!
Enter command:start 1
Do you really want to start the client with the name bcast_bot:yes
Enter password for this client:**********************
Enter command:
```

For assistance in starting and running the BroadcastBot Docker image please contact wickr-support@amazon.com.

Broadcast Bot Enterprise Installation

This section describes the requirements and configuration of the BroadcastBot for Wickr Enterprise networks.

Requirements:

Before you can install and configure the BroadcastBot, you will need to create a Wickr Bot user via the appropriate Wickr admin console, in a specific network. The associated Wickr username and password will be used when creating the BroadcastBot bot. the section called “Step 1: Create a bot user” describes how to create a Wickr IO client on the Wickr Admin console.

When running a BroadcastBot in a Wickr enterprise network you will need to download the appropriate Wickr configuration file. This file will need to be copied to the system where the BroadcastBot docker image is running. The Configuration section below identifies the location this configuration file is to be placed.

The following is a list of tokens that are required to start/configure a Wickr Enterprise BroadcastBot integration:

- WICKRIO_BOT_NAME - The name of the BroadcastBot Wickr client. This value should be automatically set.
- DATABASE_ENCRYPTION_CHOICE - Identifies if you want to encrypt the BroadcastBot information. Choices are 'yes' or 'no'.
- DATABASE_ENCRYPTION_KEY - A random sequence of bytes used to encrypt the BroadcastBot information. This string must be at least 16 characters in length. Entering a value with less than 16 characters will not encrypt the BroadcastBot information. It is not required to be encrypted but is highly recommended.
- ADMINISTRATORS - A comma separated list of Wickr users that can use the BroadcastBot to send broadcast messages. These are the only users that can use the BroadcastBot. Additional admin users can be added via the /admin command.
- WEB_INTERFACE - Identifies if you want to use the Web Interface and/or the REST API interface. Choices are 'yes' or 'no'.
- WEB_INTERFACE_PORT - The port number to use for the Web Interface and REST API interface. This value is required if WEB_INTERFACE is 'yes'.
- WEB_INTERFACE_SSL - Identifies if you want to use SSL for the Web Interface and REST API interface. Choices are 'yes' or 'no'.

- WEB_INTERFACE_SSL_KEY_LOCATION - The location of the SSL key file. This value is required if WEB_INTERFACE_SSL is 'yes'.
- WEB_INTERFACE_SSL_CERT_LOCATION - The location of the SSL certificate file. This value is required if WEB_INTERFACE_SSL is 'yes'.
- API_AUTH_TOKEN - A random sequence of bytes used as the API authorization token. This value is required if WEB_INTERFACE is 'yes'.
- HTTPS_CHOICE - Identifies if you want to use HTTPS for the Web Interface. Choices are 'yes' or 'no'.
- GOOGLE_MAPS_API_KEY - Your Google Maps API key. This is required if you want to use the /map command.
- BROADCAST_ENABLED - Enter 'yes' to enable the /broadcast command and allow broadcasts to the whole network and security groups. Enter 'no' to only allow sending to files with the /send command.

Configuration:

You will need to have some familiarity with Docker in order to configure and start the Wickr BroadcastBot. the section called “Integration setup” has some helpful information on working with Wickr Docker images.

When running in a Wickr enterprise environment you will need to get a Wickr configuration file and place it on the system where you will be running the BroadcastBot. The location must be visible to the software running on the WickrIO docker container, this is identified by the shared location option (-v) assigned by the "docker run" command. This configuration file must be present when you create the BroadcastBot. We recommend the location of where to place the Wickr configuration file is the following location, where "WickrIOShare" is the location used with the docker run command:

```
/WickrIOShare/wickr-enterprise.conf
```

The following are steps you can use to create a Wickr Enterprise BroadcastBot integration:

1. Start the Wickr IO Docker image.
2. Enter the add command at the prompt, filling in the username and password you created in step one.
3. Enable autologin.

4. Enter the broadcast bot integration from the list, for example `wickrio-broadcast-bot`
5. Respond to the configuration prompts with appropriate values, see the sample output below.
6. The broadcast bot is configured now. Start the client by entering `start` and then `y` and then the password.

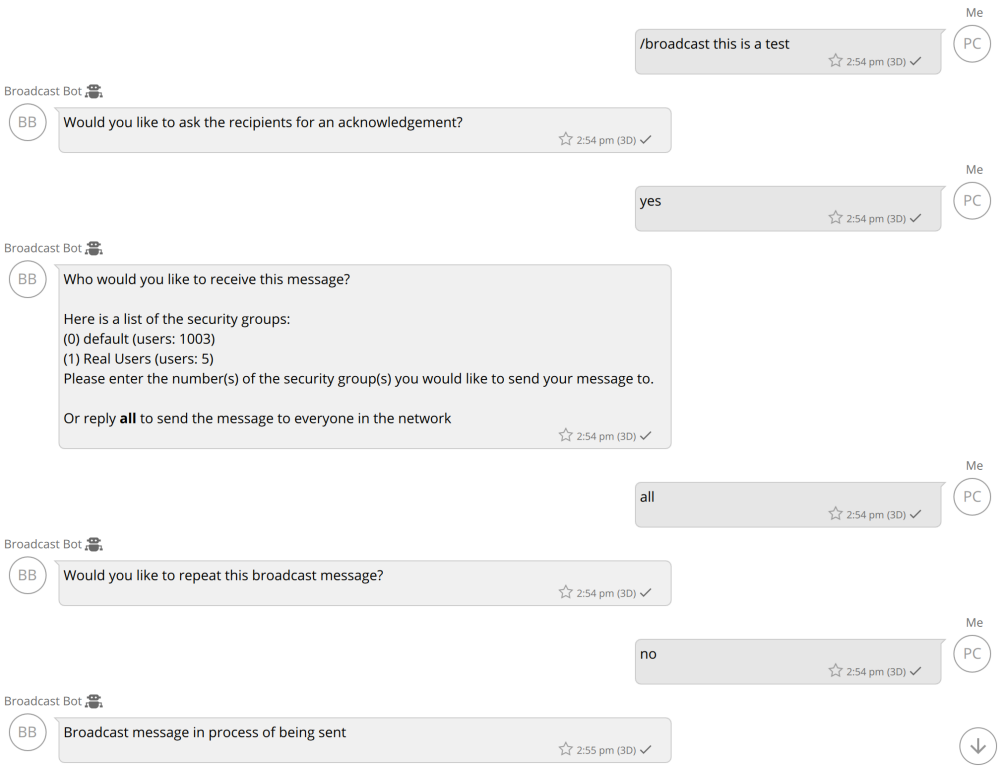
If you have followed along so far you now have the broadcast bot running on your network!

For assistance in starting and running the BroadcastBot Docker image please contact wickr-support@amazon.com.

BroadcastBot Wickr Interface

You can interact with the BroadcastBot using commands sent to the BroadcastBot over a 1to1 Wickr conversation. When administrators are being used only those approved Wickr users will have access to all of the BroadcastBot commands, without that setting all users will have full access to the functionality of the BroadcastBot. There are a small set of commands that all Wickr users can use regardless of whether administrators are being used (like the /ack command). Approved users are initially identified when the BroadcastBot is installed and configured, which is described in the Configuration section below. Additional approved users can be added or removed by any approved user on a 1on1 conversation to the BroadcastBot, using /admin commands.

To use the BroadcastBot you will interact via a 1on1 conversation with the BroadcastBot Wickr client. The BroadcastBot will prompt you for any necessary information related to what you are going to broadcast. The following shows a sample dialog with the BroadcastBot to send a broadcast message:



The BroadcastBot will send the message, file or voice memo to the destination group(s) you select. The broadcast message will be sent on 1on1 conversations between the BroadcastBot and each member of the destination group. The broadcast message will include the identity of who the broadcast was initiated by, for example:

If you want the user to acknowledge the receipt of the broadcast message that will be mentioned in the broadcast message as well. If you broadcast a file or a voice memo an additional message will be sent to include the identity of the broadcast user as well as the acknowledgement request.

Usage:

To get a list of commands available with the BroadcastBot, the /help command will present the list of the commands and a description of what each one does. The following is a list of the commands supported by the BroadcastBot, the commands in bold can only be used by approved Wickr users if administrators are being used:

Command	Description
/abort	Stops sending any remaining broadcast messages. This command is useful if you are sending to a large network or security group and need to stop the broadcast. The /status and /report statistics will indicate how many messages were aborted.
/ack	Acknowledges all messages you have received from the BroadcastBot.
/admin list	Returns a list of the admin users.
/admin add <users>	Add one or more admin users. A message will be sent to all admin users identifying the new admin user.
/admin remove <users>	Remove one or more admin users. A message will be sent to all admin users identifying the removed admin user.
/broadcast <message>	Send the text following the command to the Wickr network associate d with the BroadcastBot. The BroadcastBot will prompt you to identify who to send the message to, if you want an acknowledgement, and if you want the message to be repeated.
/cancel	This command can be entered when you are in the process of setting up a broadcast message. It will NOT cancel a message that is in process of being sent.
/delete	This command can be used to delete a file that was previously made available for the /send command.
/files	Returns a list of saved files that are available for the /send command. You can also select a file from the list and have a copy of that file sent to you.
/help	Returns a list of commands and information on how to interact with the BroadcastBot
/map	Displays a Google Maps link that shows a map that contains pins for the user locations for a specific broadcast message. Users must send

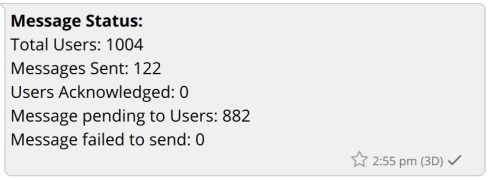
Command	Description
	their location to the broadcast bot in order for them to be included in the map.
/panel	Displays the link and token to the web user interface. This command is available only if the web application was set up during the configura tion process.
/report	Retrieves a detailed report identifying the list of users a broadcast was to be sent to. The report identifies the state of the message for that user including if the user acknowledged it. Status values include: pending, sent, failed or acknowledged. If a message failed to be sent to a user there will be a message to indicate the failure.
/send <message>	Send the text following the command to a predetermined list of users. The list of users will come from one of the files that has been uploaded to the bot and made available for this command during the file upload process.
/start	Start a new broadcast
/status	Return summary statistics associated with a broadcast.
<file>	To broadcast a file, send a file to the BroadcastBot, answer the sequence of questions and the file will be broadcast to the designated users.
<voice memo>	To broadcast a voice memo, send a voice memo to the BroadcastBot, answer the sequence of questions and the file will be broadcast to the designated users.
/version	Returns the version of your Wickr integration.

When you broadcast a file or a voice memo, the BroadcastBot will then prompt you to identify the destinations of the broadcast (security groups or network).

When you broadcast a text-based message the BroadcastBot will ask you if you want to send the message multiple times and how many minutes between each iteration of sending the broadcast.

Currently, you are allowed to wait 5, 10 or 15 minutes between each iteration of sending the broadcast.

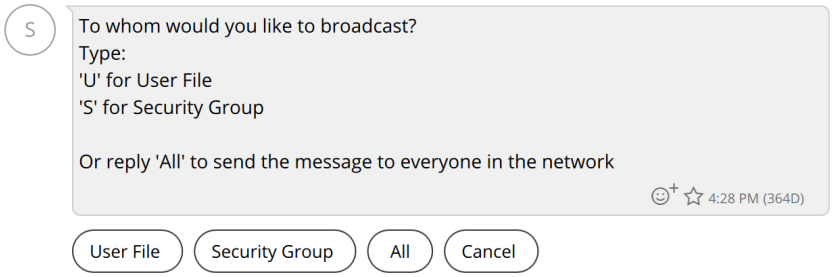
Detailed reports are returned in a CSV format, which can be imported into programs such as Excel or Pages. The following image shows a sample summary of a broadcast message, as well as the types of status information maintained by the BroadcastBot:



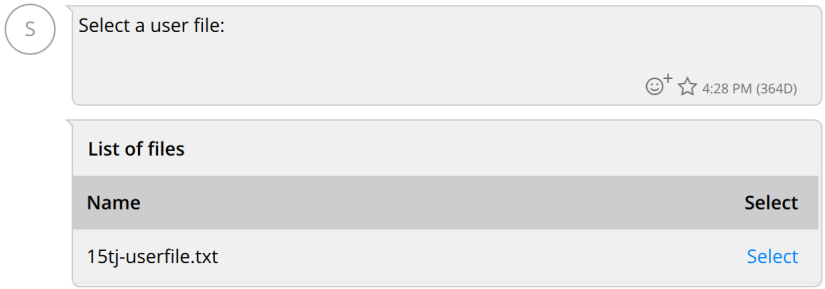
> **Note**
>
> The current broadcast bot does not support commands via a room or group conversation.

/start Command

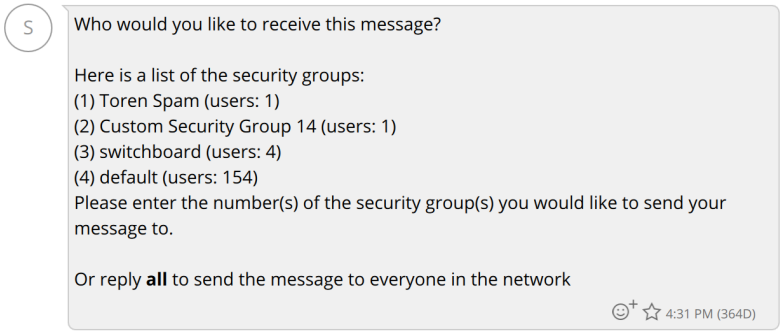
The start command is used to initiate a broadcast. When a broadcast is initiated the user will be asked recipients of the broadcast. This can be configured using a User File (a .txt file of Wickr usernames), or by stating the Security Group(s) to which this broadcast will go out.



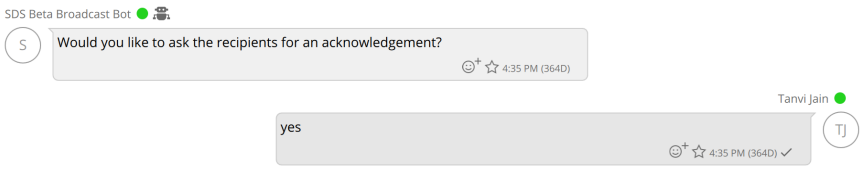
If the user selects User File, they can choose from a list of previously uploaded user files or upload a new one by hitting the '+' sign on the navigation bar:



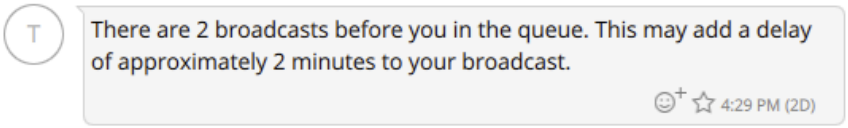
If the user selects Network or All, they have the option to broadcast the message to the whole network or to one or more security groups:



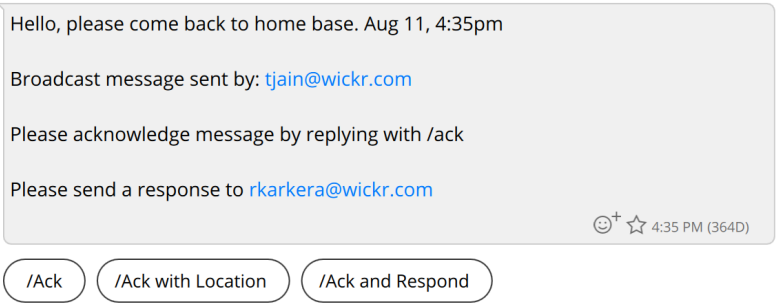
The sender can configure the broadcast to request that recipients acknowledge the message, acknowledge with their location or acknowledge with a Response.



If there are other broadcasts in the queue before the user, they will get a message with an estimated wait time:



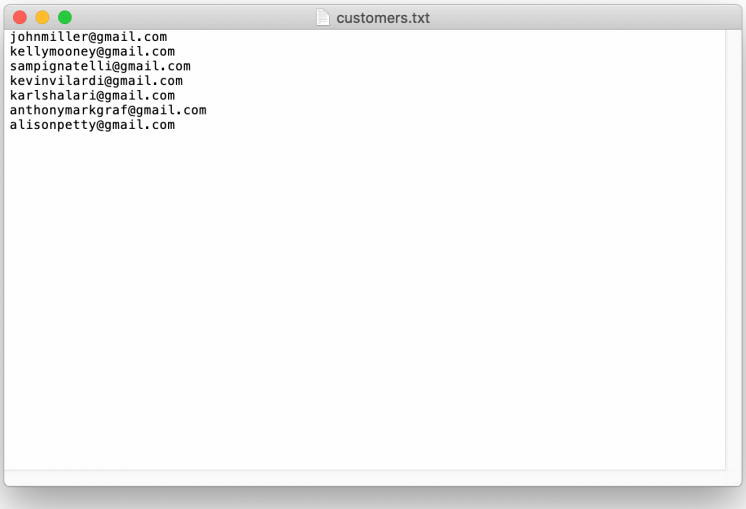
Once the user's broadcast is sent, they will receive a confirmation of this send and a status message to indicate the status of the broadcast. At any point the user can see the status of their broadcast using the /status command or the user can generate a report by running the /report command:



/send Command

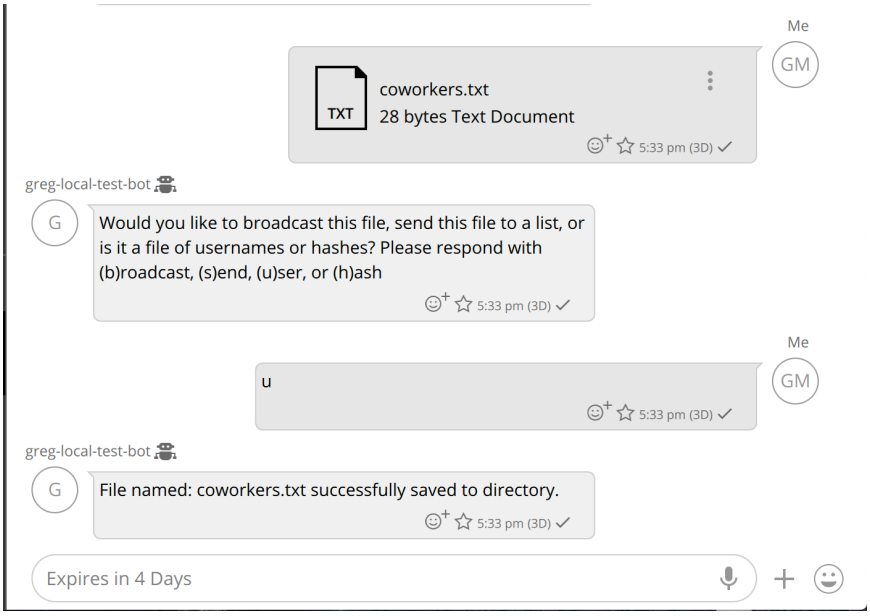
The send command is used to send a message to a predetermined list of users in your Wickr Network. These users will recieve the message in a 1 on 1 conversation with the broadcast bot. In order to use this command you must first upload a file of Wickr user ID's and make that file available for the bot and the /send command.

The file format is simple. Just write the Wickr user ID of each user you wish to send a message to on its own line in a text file (.txt). Commas or any other delimiters at the end of each line should not be included. An example is shown below:

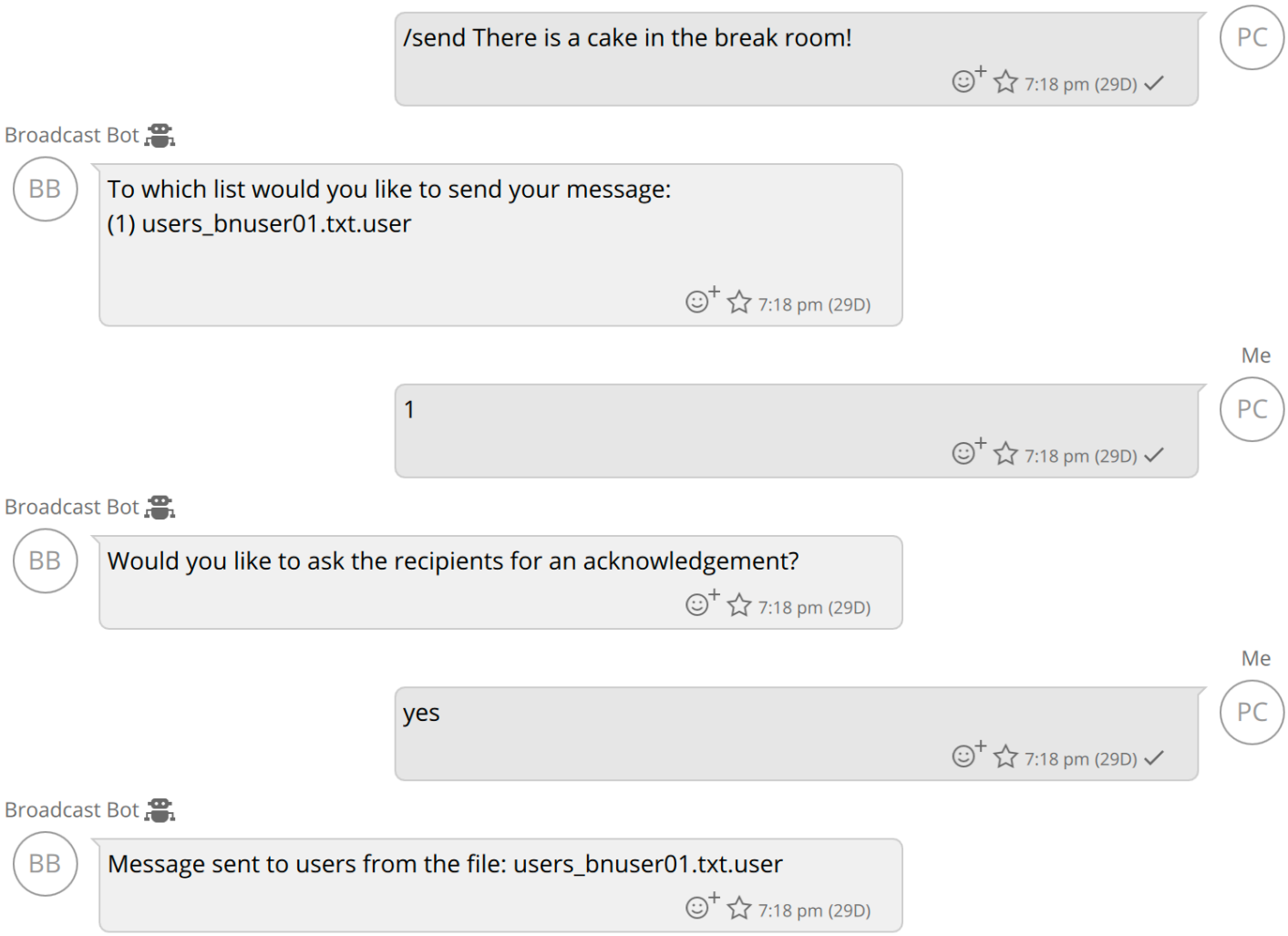


Once you have a text file of users, you are ready to upload it and make it available to the broadcast bot and send command. To do this go to your direct message conversation with the broadcast bot and click on the plus sign in the bottom right hand corner. A menu will pop up. Click on the option that says "choose file."

You will then be asked what you want to do with the file. Respond by pressing the 'u' key and then pressing enter. If you've been following along then you have just uploaded your file to the bot and made it available for the send command.



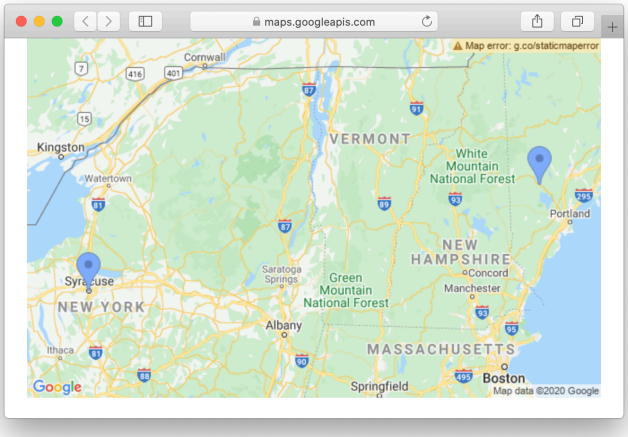
Now you are ready to send a message. To begin the process type /send followed by the message you want to send. For example if you wanted to tell people there is cake in the break room you would type the following: "/send There is cake in the break room!" You will then be prompted to choose which file of users you want to send to. Next you will be prompted if you want an acknowledgement response. Once you make a selection the message will be sent to each user and you will recieve a status message(s) detailing the progress of the message.



/map Command

This command was added in the 5.60 release of the broadcast bot.

The /map command will display Google Maps link that will display a map that contains pins for all of the users associated with a specific broadcast message. Only the users that send their location to the broadcast bot will be shown on the map. The following image shows the Google Maps image where two users sent their location to the broadcast bot:



The /map command will only work if it has been configured properly with a Google Maps API key. The following shows the prompts for the config entries associated with the /map command setup.

```
Do you want to map users locations when you send broadcasts [yes/no]: (no) :yes
Please enter your google maps api key: (false) :AIzaSyCkPH2u5f5GAOXabcdef-aJEJFjeaisl
```

When a user receives a broadcast message they can reply with their location. If the user received multiple broadcast messages and wants to share their location, they can only respond to the most recent one and that location will be used for all of the proceeding broadcasts.

Let's look at an example where a broadcast is sent out to a group of users in the morning three days in a row. The users in this group do alot of travelling during these three days. User A is in Washington DC the first day, on day two he is in New York City and day three is in Boston. Each day when he receives the broadcast, he sends his location. User B is also travelling with user A to the same locations, but he does not send his location until he reaches Boston. At the end of the three days, when you retrieve the map associated with day one you will see User A in Washington DC and User B in Boston. The map associated with day two will show User A in New York City and User B in Boston. The map associated with day three will show both users in Boston.

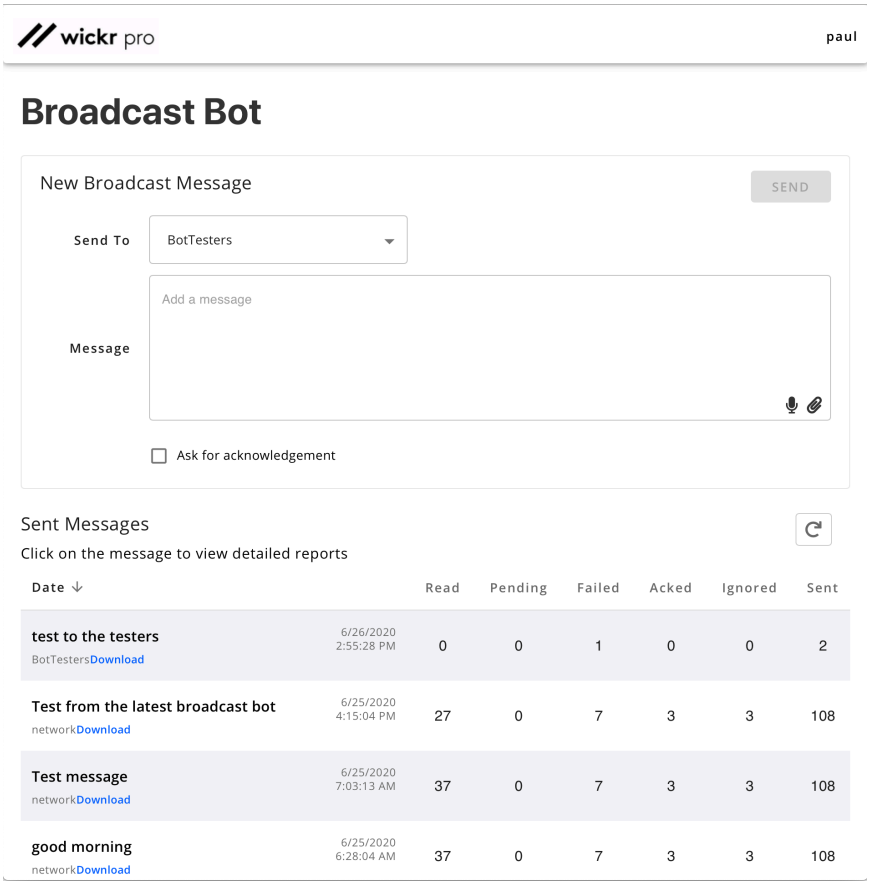
BroadcastBot Web Interface

This section describes the web interface supported by the BroadcastBot. The web interface is a browser-based interface to interact with the BroadcastBot. Use the /panel command on the BroadcastBot's Wickr interface. The /panel command will return a link that you can use to bring up

the web interface. WARNING: the link you receive is only valid for a limited time. If the link expires you will not be able to access the web interface you will have to get a new link using the /panel command.

NOTE: The BroadcastBot's web interface is initially supported as of version 5.56.

The screenshot below shows the main landing page for the BroadcastBot's web interface. On this page you can send a broadcast message and view the list of broadcast messages that you have already sent.



To begin you will want to select the recipients for the broadcast message. Using the drop down box that says "Select Security Groups" you can choose which security group will receive your broadcast message. You also have the option to broadcast to everyone in your network by selecting the "Whole Network" option.

Next write the content of your message in the text box. By clicking on the paperclip in the lower right hand corner of the text box you can attach a file to your broadcast message. You also have the option to ask each recipient for an acknowledgment of your broadcast message. To enable this feature simply check the box next to the words "Ask for acknowledgement". Once your message

is ready, click on the "Send" button in the upper right hand corner to begin broadcasting your message.

The list of sent broadcast messages will show information about the broadcast including at least the following:

- Starting text from the message sent, or the name of the file sent.
- Name of the security group or "network" (if sent to the entire network) that the message was broadcast to.
- Date and time the message was sent.
- The number of Wickr users that read the message. The Read count is only supported if read receipts are enabled for the security group.
- The number of users that are pending to be sent to.
- The number of users that the broadcast was not sent to because of some transmit failure or user failure. Details of the reason for the failed transmits can be seen in the message details screen. Failures are typically associated with user accounts that are in a bad state.
- The Acked count is the number of users that acknowledged the message. Messages are acknowledged by the /ack command or sending your location to the broadcast bot.
- The Ignored count is the number of users that were not broadcast to. If a bot is in the destination security group or network, they will not be broadcast to and will account for the ignored value.
- The Sent count is the number of users the message was sent to without receiving a send failure. This does not account for any receive failures that may have occurred.

If you click on the message text of any of the sent messages the detailed list of the users associated with the broadcast will be displayed. You will see the same statistics that were shown on the main screen as well as details of the transmission to each user in the list of users the broadcast is associated with. You can also download a JSON report of the information displayed on this screen.

To update the statistics on your broadcasted/broadcasting messages click on the refresh button in the top right hand corner of the "Sent Messages" table. Messages can also be sorted by date by clicking on the arrow next to the "Date" column header. If the arrow is pointing up messages in the table will be sorted from earliest sent to latest sent. If the arrow is pointing down messages in the table will be sorted from latest sent to earliest sent.

BroadcastBot REST API

This section describes the REST APIs supported by the BroadcastBot. This capability is initially supported in version 5.56 of the BroadcastBot. The following table identifies each of the actions the API supports, the type of HTTP request and the URL used.

API	HTTP	URL
Get security groups	GET	https://<host>:<port>/WickrIO/V2/Apps/Broadcast/<API Key>/SecGroups
Broadcast a message or file	POST	https://<host>:<port>/WickrIO/V2/Apps/Broadcast/<API Key>/Broadcast
Get list of messages sent	GET	https://<host>:<port>/WickrIO/V2/Apps/Broadcast/<API Key>/Messages?page=<Page>&limit=<Limit>
Get broadcast summary	GET	https://<host>:<port>/WickrIO/V2/Apps/Broadcast/<API Key>/Summary?messageID=<MessageID>
Get broadcast details	GET	https://<host>:<port>/WickrIO/V2/Apps/Broadcast/<API Key>/Report?messageID=<MessageID>&page=<Page>&limit=<Limit>&filter=<filter>
Abort broadcast message	POST	https://<host>:<port>/WickrIO/V2/Apps/Broadcast/<API Key>/Abort?messageID=<MessageID>
Set Event URL Callback	POST	https://<host>:<port>/WickrIO/V2/Apps/Broadcast/<API Key>/EventRecvCallback?callbackurl=<url>
Get Event URL Callback	GET	https://<host>:<port>/WickrIO/V2/Apps/Broadcast/<API Key>/EventRecvCallback
Delete Event URL Callback	DELETE	https://<host>:<port>/WickrIO/V2/Apps/Broadcast/<API Key>/EventRecvCallback

The <API Key> value is the value you entered during the configuration of the Wickr Web Interface integration.

For all of the BroadcastBot REST APIs, the "Authorization" HTTP header will have a value that is "Basic " followed by the base64 encoded value of the Authorization Token value created when the BroadcastBot is configured. For example, the Authorization Token value of "The big red fox" encodes to "VGhlIGJpZyByZWQgZm94" in base64, so you would send the string "Basic VGhlIGJpZyByZWQgZm94" in the "Authorization" HTTP header.

Get Security Groups API

This API is used to get the list of security groups that can be used to broadcast to.

```
https://<host>:<port>/WickrIO/V2/Apps/Broadcast/<API Key>/SecGroups
```

The get security groups REST API is an HTTP GET command.

The response will return a JSON array of security group information containing the ID and name of the security groups that you have access to. The following is an example of a response to the get security groups REST API:

```
[
    { "id": "2jOtbNpA", "name": "only bob" },
    { "id": "h-R3zBuV", "name": "Custom Security Group 5" },
    { "id": "hgQfaqbM", "name": "default" },
    { "id": "iNc4VjAx", "name": "Real Users" },
    { "id": "jBfxwk5u", "name": "Empty Security Group" }
]
```

> **Note**
>
> The ID value returned is the value that will be used in the broadcasting APIs. The name value is for display purposes only.

Broadcast a Message or File API

This API can be used to broadcast a message or a file. The destination for this broadcast can be either a security group, the entire network or a list of users. If you want to send to a security group, you will include the security group ID in the "security_group" value. To send to the entire network you will not send the "security_group" and "users" object in the request. To send to a list of users you will send the "users" object with the list of users. Samples are below.

This is the endpoint associated with the API:

```
https://<host>:<port>/WickrIO/V2/Apps/Broadcast/<API Key>/Broadcast
```

The broadcast message or file API is an HTTP POST command.

The body of the POST request is JSON and the contents of that JSON will determine the type of broadcast message to send as well as who the broadcast message is going to be sent to. There are several JSON fields used by this API and depending on the type of message being sent. The following table lists all of the JSON fields that are supported by the Send Message API.

KEY	Description
message	String value that is the message to be broadcast. This value is required.
security_group	This is the security group ID to send to. Do not include this entry if sending to the entire network.
users	This is a JSON list of users to send to. Do not include this entry if sending to the entire network or a security group. The object for each user can also include a "meta" object that will be associated with the user's entry for the broadcast message. See the examples below that show different uses of the "meta" object.
acknowledge	Indicates if an acknowledgement request should be part of the broadcast message. If this is "true" then the acknowledgement request will be included in the broadcast message. This value is optional, if not included an acknowledgement is not requested.
repeat_num	The number of times to repeat the broadcast message. This value is optional, if not included then the message will not be repeated.
freq_num	The number of minutes to wait between repeating a broadcast message. This value is optional but is required if the "repeat_num" value is present.
bor	The burn-on-read value to use when sending the message. This value is optional, will default to the current value set for the conversation.

KEY	Description
ttl	The time-to-live value to use when sending the message (referred to as Expiration Time in clients). This value is optional, will default to the current value set for the conversation.
user_meta	This is a boolean value (true or false) that indicates whether the meta data associated with each user should be included in the broadcast message status information.

Broadcasting Messages

To broadcast a message, the HTTP Header must include the Content-Type value of "application/json". The following is a sample JSON object to send a broadcast to the "only bob" security group from the get security group API description above. The security group ID for the "only bob" security group is included in the "security_group" JSON object:

```
{
    "security_group": "2jOtbNpA",
    "message": "Welcome to Wickr! This message will self-destruct in 5 seconds."
}
```

The following is a sample JSON object to broadcast a message to the entire network:

```
{ "message" : "This is a broadcast to a group of users",
}
```

The following is a sample JSON object to broadcast a message to a list of users:

```
{ "message" : "This is a broadcast to a group of users",
  "users" : [{"name" : "user1@company.com"},
             {"name" : "biguser44@company.com"},
             {"name" : "smalluser232@company.com"},
             {"name" : "myuser@company.com"},
             {"name" : "thelastuser@company.com"}]
}
```

The following is a sample JSON object to broadcast a message to a list of users and include meta data for each of the users:

```
{ "message" : "This is a broadcast to a group of users",
  "users" : [{"name" : "user1@company.com", "meta" : "this is data for User 1 AKA Bit
 User" },
             {"name" : "biguser44@company.com", "meta" : {"lang":"en", "id":1122324} },
             {"name" : "smalluser232@company.com", "meta" : {"lang":"en",
 "id":123243} },
             {"name" : "myuser@company.com"},
             {"name" : "thelastuser@company.com", "meta" : "Me" }],
  "user_meta" : true
}
```

The value of the "meta" object can be a JSON object or a string. When you perform a report command, the value of the "meta" object will be returned for each user in the response.

The response will be a normal HTTP 200 status if successful, or an HTTP error depending on the failure that occurred. The broadcast will go through several stages before the actual broadcast begins, specifically setting up the tables for the broadcast message status information as well as preparing the user information for the users to be broadcast to. Depending on the number of users being broadcast to this pre-broadcast process may take a few minutes (typical when sending to thousands of users).

If the broadcast is initiated successfully a JSON response will be sent containing information associated with the broadcast, including the message sent, the messageID associated with the message and the identity of the destination of the broadcast.

The following is a sample of the output for a broadcast to the entire network:

```
{
    "data": {
        "message": "This is a broadcast test",
        "message_id": "54",
        "securityGroups": []
    }
}
```

The following is a sample of the output for a broadcast to a list of users:

```
{
    "data":{
        "message":"This is a broadcast test and another test",
        "message_id":"56",
```

```
        "users":["user1@company.com"]
    }
}
```

You can use the "message_id" value returned for subsequent calls to get the broadcast message's status and report.

Broadcasting Files

To broadcast a file, the Content-Type HTTP header used for this REST API is different because it has a value of 'multipart/form-data'. This is necessary so the file can be part of the request. The body will have two parts, one for the file to broadcast, and the other part to identify who to broadcast to and if there are other broadcast settings (similar to the Broadcast Message API).

The part that identifies the file to transmit has a key value of "attachment", and the value is the actual file to transmit.

The part that contains the broadcast information is a JSON string that is exactly the same as that defined above. The "message" object is used to send a message that is also broadcast with the file.

The response will be a normal HTTP 200 status if successful, or an HTTP error depending on the failure that occurred. The broadcast will go through several stages before the actual broadcast begins, specifically setting up the tables for the broadcast message status information as well as preparing the user information for the users to be broadcast to. Depending on the number of users being broadcast to this pre-broadcast process may take a few minutes (typical when sending to thousands of users).

Get List of Messages Sent API

This API is used to get a list of the broadcast messages that were sent via the REST application APIs.

```
https://<host>:<port>/WickrIO/V2/Apps/Broadcast/<API Key>/Messages?
page=<Page>&limit=<Limit>
```

The get list of messages sent API is an HTTP GET command.

The number of broadcasts sent can grow over time so you will have to limit the size of the response by selecting an appropriate "Page" and "Limit" values. The "Page" value starts at 0. The "Limit" value identifies the number of message entries in each page. A page size of 1000 is likely to be okay. If you have more than 1000 broadcast messages, then you will have to make multiple calls to

page through the messages. Each response will contain a "max_entries" value which indicates the total number of broadcast messages associated with the Wickr user.

The response to the get list of messages sent API will look like the following example:


```
{
    "list": [
        {
            "message": "Hello. This is a broadcast to a list of users",
            "message_id": "3",
            "sender": "user123@wickr.com",
            "target": "USERS",
            "when_sent": "2020-06-08T21:18:15.511Z"
        },
        {
            "message": "Hello this is a sample broadcast to a security group",
            "message_id": "4",
            "sender": "user123@wickr.com",
            "target": "hgQfa8TM",
            "when_sent": "2020-06-08T22:08:59.855Z"
        },
        {
            "message": "This was a broadcast to an entire network",
            "message_id": "5",
            "sender": "user123@wickr.com",
            "target": "NETWORK",
            "when_sent": "2020-06-08T22:16:20.463Z"
        },
        {
            "message": "This was a broadcast of a file",
            "message_id": "13",
            "sender": "user123@wickr.com",
            "target": "NETWORK",
            "when_sent": "2020-06-09T22:17:48.469Z"
        }
    ],
    "max_entries": 5,
    "source": "user123@wickr.com"
}
```


The "max_entries" value identifies how many broadcast entries are associated with the "source" user. This value can be used to help identify how many pages of messages you may have to download.

The "target" value in each of the "list" entries identifies if the broadcast was to a specific security group, the entire network ("NETWORK") or a list of users ("USERS").

Get Broadcast Summary API

This API is used to get the summary status of a specific broadcast message. The message is identified by the message ID value, use the Get List of Messages Sent API to get the appropriate message ID.

```
https://<host>:<port>/WickrIO/V2/Apps/Broadcast/<API Key>/Summary?messageID=<MessageID>
```

The get broadcast summary API is an HTTP GET command.

This API will return a JSON object with the following values:

KEY	Description
aborted	Number of users that were not sent to because the message was aborted. See the Abort REST API for details.
acked	Number of users that acknowledged receiving this message using the /ack command.
failed	Number of users that did not get the message due to a failure to send the message. See the detail report for details on failure types.
ignored	Number of users that were not sent to because the account was a bot. Bots will not be broadcast to.
num2send	Number of users that are associated with this message.
pending	Number of users that have not been sent to yet.
read	Number of users that have read the message sent to them. This will only be set if read receipts are supported by your Wickr servers.
sent	Number of users that have been sent to for this broadcast.
status	A status value that will be a value of "Preparing" if the broadcast is in the preparation state, pending the actual broadcast of messages.

The response to this API will include all of these values. The "status" value will only be included if the broadcast is preparing the broadcast data structures. For large broadcasts this may take some time. When in this state the other status values will have a value of 0.

Get Broadcast Details API

This API is used to get the detailed status of a specific broadcast message. The information returned is a list of entries, one for each user the broadcast was transmitted to. Use the Get List of Messages Sent API to get the appropriate message ID. Since it is possible the broadcast will be sent to thousands of users your request will have to identify the page and page size values. It is recommended to not make the page size too large. This API supports a filter option to select one or more status types to retrieve for the associated MessageID.

```
https://<host>:<port>/WickrIO/V2/Apps/Broadcast/<API Key>/Report?
messageID=<MessageID>&page=<Page>&limit=<Limit>&filter=<Filter>
```

The get broadcast details API is an HTTP GET command.

The filter value is used to select which status entries to retrieve. The <Filter> value is a comma separated list of the filters to retrieve. The status values that you can use to filter on are: "pending" "sent" "failed" "acked" "ignored" "aborted" "read". This filter is optional, if you do not want to filter then do not include the filter parameter.

The response to this request will be a list of entries with the following fields:

KEY	Description
user	The Wickr ID of the user associated with this entry
status	This is the status of the broadcast to this specific user. This value can be one of the values shown in the Get Broadcast Summary API section above.
statusMessage	This is a message associated with the status value. If the status is "failed" this field will identify why the send failed. If the user acknowled ges the message by sending their location, this field will have a URL that shows their location.
sentDate	This is the date and time the message was sent. The date format is the following: 2020-06-14 17:27:27 UTC

KEY	Description
readDate	This is the date and time the message was read, if read receipts are supported by your Wickr server. The date format is the following: 2020-06-14 17:27:27 UTC

Abort Broadcast Message API

This API is used to abort a broadcast message that is currently being transmitted. This is typically only useful for broadcasts to large numbers of users. Broadcasts to small numbers of users will likely complete too quickly for this API to be useful. Use the Get List of Messages Sent API to get the appropriate message ID.

```
https://<host>:<port>/WickrIO/V2/Apps/Broadcast/<API Key>/Abort?messageID=<MessageID>
```

The abort broadcast message API is an HTTP POST command.

The response will be a normal HTTP 200 status if successful, or an HTTP error depending on the failure that occurred. The response body will contain a JSON object with a "result" string and a "status" object that contains the summary status associated with the MessageID used to perform the abort operation. The following is a sample output from a successful abort command:

```
{
    "result": "Success",
    "status": "{
        "aborted": 1900,
        "acked": 0,
        "failed": 0,
        "ignored": 0,
        "num2send": 2000,
        "pending": 0,
        "read": 0,
        "sent": 100
    }
}
```

Abort commands that fail will return a string identifying the type of error. Typical errors invalid messageID values or trying to abort a messageID that is not associated with the bot user.

Event Callbacks

Event callbacks will define a URL that the bot client will connect to when a messaging event occurs. Information about the event will be sent to the defined URL. It is assumed that there is an application consuming, and acknowledging, events that are pushed to this URL. For example, if you run a process on the same machine as the Wickr IO client you can use a URL like the following:

```
http://localhost:4100
```

Events that are posted to the set URL will have the following sample format:

```
{
    "message_id" : "9",
    "reason" : "Failed verification",
    "user" : "user555@somewher.com"
}
```

Set Event Callback URL API

You will need to configure the specific URL that the Wickr IO client will send events to. To set the URL callback value, send an HTTP POST request using the following URI:

```
https://<host>:<port>/WickrIO/V2/Apps/Broadcast/<API Key>/EventRecvCallback?
callbackurl=<url>
```

The client will respond back with a success or failure response. Any events that occur after this call is performed will be sent to the defined URL.

Get Event Callback URL API

To get the currently set URL callback send an HTTP GET request using the following URI:

```
https://<host>:<port>/WickrIO/V2/Apps/Broadcast/<API Key>/EventRecvCallback
```

The client will respond with either a success (200) or failure response. If there is a URL callback defined the following is the format of the body:

```
{
```

```
    "callbackurl": "https://localhost:4008"
}
```

Delete Event Callback URL API

If a URL callback is no longer needed, then the delete URL callback API should be used. To delete a URL callback, send an HTTP DELETE request using the following URI:

```
https://<host>:<port>/WickrIO/V2/Apps/Broadcast/<API Key>/EventRecvCallback
```

The client will respond with a success or failure response.

Broadcastbot Message Send Failures

It is possible that when broadcasting to a group of users there may be a failure with the destination user's account. This section will describe some of the possible failures that may happen when sending a broadcast.

KEY	Description
"Failed verification"	This failure message indicates the account is not valid. Could be in a suspended state as well.
"Could not find user record"	This failure message also indicates the account is not valid. Could be in a suspended state as well.
"Message failed to send, check user account"	This failure usually indicates an issue with the user account being sent to. Something like no active devices associated with the account.
"msgSvcSend failed"	This failure is due to an internal issue. Could be the process associated with transmitting messages was not fully initialized, this could be very rare. Retransmitting the message to the effected user(s) should work.
"sendFile failed"	This failure can happen when broadcasting a file but is very rare as well. Retransmitting the message to the effected user(s) should work.
"No users on the 1to1"	This failure is an internal error and is very rare. If this type of error does occur, retransmitting should work.

There are other failures possible and they are associated with network or system issues that may occur. If those errors do occur retransmitting the message to the effected user(s) should work.

Web Interface Integration

The Web Interface integration allows remote software the ability to interact with the associated Wickr IO client via a REST API. The Wickr IO Web Interface integration supports the REST API via either an HTTP or HTTPS interface. This REST API supports the ability to create conversations, send and receive messages to other Wickr users, and perform other Wickr tasks.

The Wickr IO Web Interface integration software is located here.

REST API Configuration

During the installation of the Wickr IO Web Interface software module, you will need to configure several properties that are needed to access the URL endpoints. The Wickr IO command line interface will walk you through entering these values. The following table describes the values that you will have to enter during the configuration of the Wickr IO Web Interface:

Value	Description
Port	The TCP/IP port that the Wickr IO bot will listen on. NOTE: you will have to add the port to the `docker run` command that starts the Wickr IO Docker container so that port is made available to the Wickr IO integration software.
API-Key	The API Key that is used in every endpoint call. This is the <API Key> value that is contained in every endpoint URL, as is shown in the table in the previous section.
Basic Authorization Token	The authentication string used to generate the Base64 value to be sent in the authorization field of the HTTP Header (Recommended: 24-character alphanumeric string). You will need to generate a Base64 value of this token and the add it to the HTTP authorization header (i.e. Basic MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNA==).
SSL Key Location	Full path name of the .key file (only required if you are going to use HTTPS). The file must be located in the shared directory that the integration software running on the Docker image can access.

Value	Description
SSL Cert Location	Full path name of the .cert file (only required if you are going to use HTTPS). The file must be located in the shared directory that the integration software running on the Docker image can access.

For HTTPS and SSL support, you need an OpenSSL certificate file and a key file. Both can be created with the following command:

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout my.key -out my.cert
```

Authentication

This version of the Web Interface supports basic authentication. The authentication will use the "Authorization" HTTP header to send the necessary authentication information to the Wickr IO server. If the proper authentication information is not presented to the Wickr IO server then an HTTP 401 response will be sent.

When using basic authentication, a base64 encoded string will be sent to the Wickr IO server. The following steps should be performed for basic authentication:

1. When the associated Wickr IO bot client is configured and associated with the Web Interface integration, the associated authentication string will be setup. You will use this string to generate the base64 encoded string.
2. Base64 encode the string mentioned above.
3. Supply an "Authorization" header with content "Basic " followed by the encoded string. For example, the string "The big red fox" encodes to "VGhlIGJpZyByZWQgZm94" in base 64, so you would send the string "Basic VGhlIGJpZyByZWQgZm94" in the "Authorization" HTTP header.

> **Warning**
>
> The "Authorization" header is encoded - NOT encrypted - thus HTTP basic authentication is only effective over secure connections. Always use the Web Interface REST API over HTTPS when communicating over insecure networks.

Web Interface Integration Installation

This section shows the steps to add a Wickr IO Web Interface integration. The following steps will create a Wickr IO Bot with the Web Interface integration. This integration will expose the HTTP/HTTPS interface to facilitate access to the associated Wickr IO client. When installing the Web Interface Integration on a host machine that has multiple network interfaces you can either bind the docker image that the Web Interface Integration resides on to all interfaces on the host machine OR you can bind to a specific interface by specifying the IP address.

1. Using the admin console create the associated Wickr IO bot user by entering a display name, username, and password.
2. Make a directory for your docker volume:
   - `sudo mkdir /opt`
   - `cd /opt`
   - `sudo mkdir WickrIO`
   - For Docker running on Mac, if not added already, add '/opt/WickrIO' to shared paths from Docker -> Preferences -> File Sharing. See https://docs.docker.com/docker-for-mac/osxfs/#namespaces for more info.
3. If you are going to use HTTPS to communicate with the Wickr IO integration you will need to generate a cert and key file. For example, you can use the following command to do so:

   ```
   openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout my.key -out my.cert
   ```

   You will need to place the key and certificate files in a location the software running on the docker container can access, this is somewhere in the /opt/WickrIO directory (you created in the previous step).
4. Start the docker image, which will download the docker image if needed. Note: the name you can assign to this docker image and use for other docker commands:

   ```
   docker run -v /opt/WickrIO:/opt/WickrIO -p 4001:4001 -d --restart=always --
   name="MyWickrIOImage" -ti public.ecr.aws/x3s2s6k3/wickrio/bot-cloud:latest
   ```

   Please note the "-p 4001:4001" option which identifies which TCP port the web interface will listen on, you will have to enter that later.

If your host machine has multiple network interfaces and you would like to specify the network interface the Web Interface Integration will run on you can add the IP address of that network interface to the docker run command like so:

```
docker run -v /opt/WickrIO:/opt/WickrIO -p xx.xx.xx.xx:4001:4001 -d --restart=always
 --name="MyWickrIOImage" -ti public.ecr.aws/x3s2s6k3/wickrio/bot-cloud:latest
```

Where xx.xx.xx.xx is the IP address of the network interface you would like to use.

NOTE: If you are on Wickr GovCloud, please remember to modify the above command to replace docker image link with GovCloud image i.e `public.ecr.aws/x3s2s6k3/wickrio/bot-cloud-govcloud:latest`

5. Attach to the docker image, using the name from the previous step: `docker attach MyWickrIOImage`
6. Agree to the license agreement.
7. Enter the add command at the prompt, filling in the username and password you created in step one.
8. Enable autologin.
9. Enter the web interface integration from the list, for example `wickrio_web_interface`
10. Enter the port number you identified previously.
11. Enter an API Key value. This is a string that will be part of every REST call you make to the web interface bot.
12. Enter a Basic authorization token value. This is an alphanumeric value used as the basic authorization for the REST calls. It is recommended that you enter at least 24 characters.
13. If you are using HTTPS enter the location and name of the key and certificate files. These files will have to be located in the /opt/WickrIO directory you created earlier, so that the Wickr IO integration software running on the Wickr IO Docker image have access to the files.
14. The web interface bot is configured now. Start the client by entering `start` and then `y` and then the password.

If you have followed along so far you now have the Web Interface Bot running on your network! The REST API is fully described in the next section. To access the REST API you can use curl or a program like Postman to test that everything is working as expected.

The following is sample output from adding a web interface bot:

```
Enter command:add
Enter the user name:test_web_bot
Enter the password:********
Creating user:  "test_web_bot"

Begin registration with password.
Begin register new user context.

Begin register existing user context.

Successfully created user

Successfully logged in as new user!

Our work is done here, logging off!

Return code from provision is: 0

The autologin capability allows you to start a bot without having to enter the
password, after the initial login.
NOTE: The bot client's password is NOT saved to disk.

Do you want to use autologin? (default: yes):
Searching NPM registry
Searching NPM registry
Searching NPM registry

These integrations are local:
  - hubot

These integrations are from the NPM registry:
  - wickrio_web_interface
  - wickrio-file-bot
  - wickrio-hello-world-bot
  - wickrio-example-app

Please enter one of:
  - The full integration name from the list above
  - The word "search" to search the NPM registry for an integration
  - The word "import" to import an integration
  - The word "quit" to cancel adding the bot
```

```
Enter the bot integration to use:wickrio_web_interface
*****************************************************************
Begin setup of wickrio_web_interface software for test_web_bot
Copying wickrio_web_interface from the NPM registry
Installing wickrio_web_interface software for test_web_bot
Installing
Begin configuration of wickrio_web_interface software for test_web_bot
going to use /tmp/WickrIOSvr.WDmxyv
 Please enter your client bot's port::4001
 Please enter your client bot's API-Key::testAPIkey1234
 Please create an Web API Basic Authorization Token(we recommend an alphanumeric string
 with at least 24 characters)::12345678901234567890123 4
 Do you want to set up an HTTPS connection with the Web API Interface(Recommended)(y/
n)::y
 Please enter the name and location of your SSL .key file::/opt/WickrIO/files/my.key
 Please enter the name and location of your SSL .crt file::/opt/WickrIO/files/my.cert

Integration files written to:
/opt/WickrIO/clients/test_web_bot/integration/wickrio_web_interface

End of setup of wickrio_web_interface software for test_web_bot
*****************************************************************
Successfully added record to the database!
Enter command
```

Web Interface REST API

This section describes the REST APIs that are supported by the 2.x version of the Wickr IO Web Interface integration. The following table identifies each of the actions the API supports, the type of HTTP request and the URL used.

API	HTTP	URL
Send Message	POST	https://<host>:<port>/WickrIO/V1/Apps/<API Key>/Messages
Send File	POST	https://<host>:<port>/WickrIO/V1/Apps/<API Key>/File
Set Message URL Callback	POST	https://<host>:<port>/WickrIO/V1/Apps/<API Key>/MsgRecvCallback?callbackurl=<url>

API	HTTP	URL
Get Message URL Callback	GET	https://<host>:<port>/WickrIO/V1/Apps/<API Key>/MsgRecvCallback
Delete Message URL Callback	DELETE	https://<host>:<port>/WickrIO/V1/Apps/<API Key>/MsgRecvCallback
Get Received Messages	GET	https://<host>:<port>/WickrIO/V1/Apps/<API Key>/Messages?start=<index>&count=<number>
Get Statistics	GET	https://<host>:<port>/WickrIO/V1/Apps/<API Key>/Statistics
Clear Statistics	DELETE	https://<host>:<port>/WickrIO/V1/Apps/<API Key>/Statistics
Create Secure Room	POST	https://<host>:<port>/WickrIO/V1/Apps/<API Key>/Rooms
Get Room	GET	https://<host>:<port>/WickrIO/V1/Apps/<API Key>/Rooms/<vGroupID>
Get Rooms	GET	https://<host>:<port>/WickrIO/V1/Apps/<API Key>/Rooms
Delete Room	DELETE	https://<host>:<port>/WickrIO/V1/Apps/<API Key>/Rooms/<vGroupID>
Leave Room	DELETE	https://<host>:<port>/WickrIO/V1/Apps/<API Key>/Rooms/<vGroupID>&reason=leave
Modify Room	POST	https://<host>:<port>/WickrIO/V1/Apps/<API Key>/Rooms/<vGroupID>
Create Group Conversation	POST	https://<host>:<port>/WickrIO/V1/Apps/<API Key>/GroupConvo
Get Group Conversations	GET	https://<host>:<port>/WickrIO/V1/Apps/<API Key>/GroupConvo

API	HTTP	URL
Get Group Conversation	GET	https://<host>:<port>/WickrIO/V1/Apps/<API Key>/GroupConvo /<vGroupID>
Delete Group Conversation	DELETE	https://<host>:<port>/WickrIO/V1/Apps/<API Key>/GroupConvo/<vGroupID>

The <API Key> value is the value you entered during the configuration of the Wickr Web Interface integration.

Send Message APIs

The send message APIs are used to send a message to one or more Wickr clients. To have a specific Wickr IO client send a message, send an HTTP POST message to the following URI:

```
https://<host>:<port>/WickrIO/V1/Apps/<API Key>/Messages
```

The body of the POST request is JSON and the contents of that JSON will determine the type of Wickr message and the destination(s) of the message. You can send a message to one or more 1-to-1 recipients, a secure room or a group conversation. To send to a secure room or a group conversation, the room will have to been created before sending any messages. You will also need the vGroupID associated with the secure room or group conversation.

The send message API also supports sending files.

There are several JSON fields used by this API and depending on the type of message or file being sent. The following table lists all of the JSON fields that are supported by the Send Message API.

KEY	Description
message	String value that is the message to be sent, when sending a Wickr message. Not used when sending a file.
users	This is a list of Wickr user IDs to send a message to. Not used when sending to a secure room or group conversation, use the vgroupid value when sending to secure room or group conversations. A separate 1on1 message will be sent to each of the users in this list.

KEY	Description
bor	The burn-on-read value to use when sending the message. Optional, will default to the current value set for the conversation.
ttl	The time-to-live value to use when sending the message (referred to as Expiration Time in clients). Optional, will default to the current value set for the conversation.
vgroupid	The vgroupid to use when sending to a secure room or group conversat ion. Do NOT use this value when sending to 1on1 conversations.
runtime	Used to specify when the message(s) are to be sent. The format of this value is ISO 8601 extended format: either yyyy-MM-dd for dates or yyyy-MM-ddTHH:mm:ss (e.g. 2017-07-24T15:46:29), or with a time-zone suffix (Z for UTC otherwise an offset as [+
attachment	This is a JSON object that identifies a file to be sent. The message value will be disregarded if the attachment field is used.
attachments	Identifies files to be sent. Currently only supports one file. Future implementation will support sending multiple files.

When sending files, the location of the files either needs to be located on the Wickr IO system or addressable and downloadable using a specific URL. If the file is located on the Wickr IO system, it must be located where the Wickr IO client running on the Docker image can access it.

Sending a single message to multiple recipients

To send a message to one or more recipients, the HTTP POST message should contain a JSON body with the following format:

```
{
    "message": "Welcome to Wickr! This message will self-destruct in 5 seconds.",
    "users": [
        { "name" : "username001" },
        { "name" : "username002" }
    ]
}
```

The "users" field may contain an array of 1 or more users to send the message to. The message will be sent to each user on a separate 1-to-1 conversation. So, if the POST message contains 5 users then 5 messages will be sent, using the text from the "message" field.

Sending a message with Burn-on-Read

When sending a message, you can also set the specific burn on read (BOR) value for the message. The following format shows how to set the BOR value to 10 seconds:

```
{
    "message": "Welcome to Wickr! This message will self-destruct in 5 seconds.",
    "users": [ { "name": "username002" } ],
    "bor": 10
}
```

Sending a message to a Secure Room

If you want to send a message to a secure room or a group conversation you will need to get the vGroupID associated with the room. The vGroupID will be returned when you create the room/conversation using the appropriate API. Also, the get rooms API will return a list of known rooms that you can send to, the vGroupID is contained in the response. To send to a secure room or group ID the following is an example:

```
{
    "message": "Welcome to Wickr! This message will self-destruct in 5 seconds.",
    "vgroupid": "S8a97892379289bca97929370982271892839283749283749283 4"
}
```

Send File APIs

The Wickr IO client supports three different ways to send files.

- Sending a file that is directly accessible by the Wickr IO client.
- Sending a file that is referenced by a URL that the Wickr IO client can download and then send.
- Sending a file by uploading the file to the Wickr IO client in the REST API request.

> **Warning**
>
> Please make sure to use the correct API endpoint, since the first two methods use a different endpoint than the third method.

Send Files Residing on the Wickr IO Client

To send files that are directly accessible by the Wickr IO client you will send an HTTP POST message to the following URI:

```
https://<host>:<port>/WickrIO/V1/Apps/<API Key>/Messages
```

The body content of the request will be of "application/json" type, and will contain a list of users to send the file to, the file path name where the file is located and a display name for the file. The following is an example of sending a file to a single user, where the file is located on the Wickr IO system:

```
{
    "users": [ { "name" : "username001" } ],
    "attachment": {
        "filename" : "/opt/WickrIO/pictures/picturesent.jpg",
        "displayname" : "PictureSent.jpg"
    }
}
```

In this example, the file to be sent is already on the Wickr IO client. The filename identifies the full path to the file. The Send Message API will respond with a 202 (Accepted) response, unless there is an error.

> **Warning**
>
> Like any data your integration writes to disk, files sent or received by the Wickr IO client are decrypted and remain on the host machine until removed by you or your software.

Send Files Referenced by a URL

To send files using a URL that are accessible by the Wickr IO client, you will send an HTTP POST message to the following URI:

```
https://<host>:<port>/WickrIO/V1/Apps/<API Key>/Messages
```

The Wickr IO client will download the file first and then send the message. The following is an example of sending a file to a single user, where the file is located at an external URL example.com.

```
{
    "users": [ { "name" : "username001" } ],
    "attachment": {
        "url" : "https://example.com/pictures/picturesent.jpg",
        "displayname" : "PictureSent.jpg"
    }
}
```

You can also send files to secure rooms and group conversations using the vgroupid instead of listing the users to send to. The following is an example of sending the file to a secure room.

```
{
    "vgroupid" : "Sd740e3077714bcc0020806bc5b318a4ca766f9fe4737e6952a81bf0d9a75407",
    "attachment": {
        "url" : "https://example.com/pictures/picturesent.jpg",
        "displayname" : "PictureSent.jpg"
    }
}
```

Send Files Passed in the REST API Request

This is a new endpoint for the Web Interface integration as of the v5.60 version.

The send file API is used to send a file to one or more Wickr clients, or a specific room conversation. The previously described send message APIs still work to send files as well. To have a specific Wickr IO client send a message, send an HTTP POST message to the following URI:

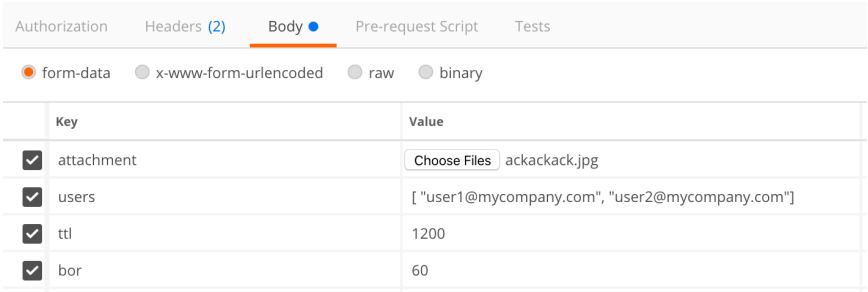
```
https://<host>:<port>/WickrIO/V1/Apps/<API Key>/File
```

You can send a file to one or more 1-to-1 recipients, a secure room or a group conversation.
To send to a secure room or a group conversation, the room will have to been created before sending any messages. You will also need the vGroupID associated with the secure room or group conversation.

The body of this POST request will use the following Key/Value pairs:

KEY	Description
attachment	This is the actual file being sent.
users	This is a JSON list of Wickr user IDs to send a message to. For example: "[user1, user2, user3]". This Key is not used when sending to a secure room or group conversation, use the vgroupid Key when sending to secure room or group conversations. A separate 1on1 message will be sent to each of the users in this list. If you want to send using the "users" key make sure the "vgroupid" key is not in the body.
bor	The burn-on-read value to use when sending the message. Optional, will default to the current value set for the conversation.
ttl	The time-to-live value to use when sending the message (referred to as Expiration Time in clients). Optional, will default to the current value set for the conversation.
vgroupid	The vgroupid to use when sending to a secure room or group conversat ion. Do NOT use this value when sending to 1on1 conversations using the "users" key.

For those familiar with Postman, the following shows an example body set to send a file to two users:



Make sure to follow the formatting shown for the "users" value.

Warning

The content-type of the body is "multipart/form-data", which includes the "attachment" and the "users" or "vgroupid". The "ttl" and "bor" are optional.

Received Message

The Wickr IO client has the capability to receive messages and the Web Interface integration provides several ways to get access to those messages. The messages can be forwarded to another application or retrieved via a REST call. To have received messages forwarded to another application a callback location will need to be set (see the URL Callbacks section). If a callback method is not defined the messages will queue up on the Wickr IO client until they are retrieved. The get received messages API can be used to retrieve the messages.

The format of the messages received are described in the section called “Wickr message formats”.

Note

If a callback destination is configured, the Wickr IO client will queue these messages to be sent to that destination. These messages will remain in this queue until they have been successfully posted to the desired destination.

URL Callbacks

This type of callback will define a URL that the client will connect to when a message is received, the received message will be sent to the URL. It is assumed that there is an application consuming messages that are pushed to this URL. For example, if you run a process on the same machine as the Wickr IO client you can use a URL like the following:

```
http://localhost:4100
```

Wickr can supply a sample program that accepts messages on a specific port and posts to a log file.

Set URL Callback API

You will need to configure the specific URL that the Wickr IO client will send incoming messages to. To set the URL callback value, send an HTTP POST request using the following URI:

```
https://<host>:<port>/WickrIO/V1/Apps/<API Key>/MsgRecvCallback?callbackurl=<url>
```

The client will respond back with a success or failure response. Any messages received after this call is performed will be sent to the defined URL.

Get URL Callback API

To get the currently set URL callback send an HTTP GET request using the following URI:

```
https://<host>:<port>/WickrIO/V1/Apps/<API Key>/MsgRecvCallback
```

The client will respond with either a success (200) or failure response. If there is a URL callback defined the following is the format of the body:

```
{
    "callbackurl": "https://localhost:4008"
}
```

Delete URL Callback

If a URL callback is no longer needed then the delete URL callback API should be used. To delete a URL callback, send an HTTP DELETE request using the following URI:

```
https://<host>:<port>/WickrIO/V1/Apps/<API Key>/MsgRecvCallback
```

The client will respond with a success or failure response.

Statistics APIs

Each Wickr IO client maintains statistics associated with the number of messages sent and received. The statistics will also include information about send and receive errors as well as the number of pending messages. This section describes the APIs associated with statistics.

Get Statistics API

This API will retrieve statistics that are saved on the client. These are a little different than the statistics that can be retrieved via the main console interface. Here is the HTTP GET used to get client statistics:

```
https://<host>:<port>/WickrIO/V1/Apps/<API Key>/Statistics
```

The following is the type of output received in the response. Some of these statistics may be missing if they have not been changed.

```
{
    "statistics": {
        "message_count": 5,
        "pending_messages": 0,
        "sent": 7,
        "received": 3,
        "sent_errors": 1,
        "recv_errors": 1
    }
}
```

The following table has a description of each of the statistics returned by this API:

Statistics	Description
message_count	The number of incoming messages that are currently on the Wickr IO client.
pending_messages	The number of messages that are to be sent from the specific Wickr IO client.
sent	The number of messages that have been sent by the Wickr IO client.
received	The number of messages that the Wickr IO client has received.
sent_errors	The number of errors that have occurred while trying to send messages.
recv_errors	The number of errors that occurred while receiving messages.
pending_callback_messages	The number of messages on the callback message queue. These are messages received by the Wickr IO client, that are waiting to be send to a callback process.

Statistics	Description
outbox_sync	The number of outbox sync messages received. These are messages that were sent by another device for this Wickr IO client.

Example curl script to perform a get statistics:

```
curl -v -k -s -X GET -H "$AUTH" http://10.2.0.1:4001/WickrIO/V1/Apps/123456/Statistics
```

Clear Statistics API

This API will clear the current statistics that are saved on the client. Use HTTP DELETE with the following URI to clear the statistics on the specific Wickr IO client:

```
https://<host>:<port>/WickrIO/V1/Apps/<API Key>/Statistics
```

Secure Room APIs

This section describes the APIs associated with secure rooms. Using these APIs you can create, modify, get, delete and leave secure rooms that the Wickr IO client is a part of.

Create Secure Room API

This API will create a new secure room. To create a secure room, send an HTTP POST command using the following URI:

```
https://<host>:<port>/WickrIO/V1/Apps/<API Key>/Rooms
```

The body of this request will contain the information associated with the room. The following is the format of the JSON data for the body of the request:

```
{
    "room": {
        "title" : "Room Title",
        "description" : "Description of the room",
        "ttl" : 3600,
        "bor" : 60,
        "members" : [
            { "name" : "username001@wickr.com" },
            { "name" : "username002@wickr.com" }
```

```
        ],
        "masters" : [
            { "name" : "username001@wickr.com" },
        ]
    }
}
```

The response will either be an error with a description of that error or a successful response with the vGroupId of the newly created secure room. The following is an example of a successful response:

```
{
    "vgroupid": "S0b503ae14cc896aad758ce48f63ac5fae0adccd78ef18cde82563c63b2c7761"
}
```

The following is a sample curl script to create a room, the data.json file contains the body of the request. The data.json contents would be something similar to that shown above.

```
curl -v -k -s -X POST -H "$AUTH" -H "$CONTENT" http://10.2.0.10:4001/WickrIO/V1/
Apps/123456/Rooms -d "@data.json"
```

Get Rooms API

This API will return a list of rooms that are known by the Wickr IO client. The Wickr IO client will only know about rooms that it is a member of. To get a list of rooms send an HTTP GET to the following URI:

```
https://<host>:<port>/WickrIO/V1/Apps/<API Key>/Rooms
```

The Wickr IO client will respond with a JSON array of secure rooms. The format of the response will look like the following:

```
{
    "rooms": [
        {
            "description": "Room description",
            "masters": [
                { "name" : "username001" }
            ],
            "members": [
                { "name" : "username001" },
```

```
                { "name" : "username002" }
            ],
            "title": "Room Title",
            "ttl": "7776000",
            "bor": "0",
            "vgroupid":
 "S00bf0ca3169bb9e7c3eba13b767bd10fcc8f41a3e34e5c54dab8bflkjdfde"
        }
    ]
}
```

The following is sample curl script to retrieve the list of rooms:

```
curl -v -k -s -X GET -H "$AUTH" http://10.2.0.20:4001/WickrIO/V1/Apps/123456/Rooms
```

Get Secure Room API

This API will return details of a specific secure room. Send an HTTP GET using the following URI:

```
https://<host>:<port>/WickrIO/V1/Apps/<API Key>/Rooms/<vGroupID>
```

The Wickr IO client will respond with a JSON structure containing information for the specified conversation. The format of the response will look like the following:

```
{
    "rooms": [
        {
            "description": "Room description",
            "masters": [
                { "name" : "username001" }
            ],
            "members": [
                { "name" : "username001" },
                { "name" : "username002" }
            ],
            "title": "Room Title",
            "ttl": "-1",
            "vgroupid":
 "S00bf0ca3169bb9e7c3eba13b767bd10fcc8f41a3e34e5c54dab8bflkjdfde"
        }
    ]
}
```

Delete Room API

In order to delete a secure room, you will need to have the vGroupID associated with that room. You can use the get rooms API to get the list of rooms known by the Wickr IO client, then determine which room to delete. Also, saving the vGroupID returned from the create room API can be used as well.

To delete a secure room, send an HTTP DELETE command using the following URI:

```
https://<host>:<port>/WickrIO/V1/Apps/<API Key>/Rooms/<vGroupID>?reason=delete
```

The secure room with the same vGroupID will be deleted. This API is used for both the delete room and the leave room action, the `reason` argument identifies which action to perform. The default value for the `reason` argument is the delete action, in which case the `reason` argument can be omitted.

The following curl script is an example of how to delete a specific room:

```
curl -v -k -s -X DELETE -H "$AUTH"  http://10.100.8.27:6379/WickrIO/V1/Apps/123456/
Rooms/S3947c067fa3edc9b0154e82e9ed1cf39904784f344e5923c4c683f27bed2faf
```

Leave Room API

In order to leave a secure room, you will need to have the vGroupID associated with that room. You can use the get rooms API to get the list of rooms known by the Wickr IO client, then determine which room to leave. Also, saving the vGroupID returned from the create room API can be used as well.

To leave a secure room, send an HTTP DELETE command using the following URI, make sure to specify the `reason` argument with the value of `leave`.

```
https://<host>:<port>/WickrIO/V1/Apps/<API Key>/Rooms/<vGroupID>?reason=leave
```

Modify Room API

This API is used to modify some of the settings associated with a secure room. The following secure room attributes can be modified using this API:

- TTL
- BOR

- Description
- Title
- Members
- Moderators

To modify any of these values for a secure room send an HTTP POST command using the following URI:

```
https://<host>:<port>/WickrIO/V1/Apps/<API Key>/Rooms/<vGroupID>
```

The body of the request will identify the values to change and the new values to assign. The following JSON is an example of a body for the modify room API which will set the BOR and TTL values:

```
{
    "ttl": 66000,
    "bor": 300
}
```

Group Conversation APIs

This section describes the APIs associated with group conversations. Using these APIs you can create, get or delete group conversations that the client is a part of.

> **Note**
>
> These APIs are only available in versions 4.35 and newer of the Wickr IO client.

Create Group Conversation API

This API will create a new group conversation. To create a group conversation, send an HTTP POST command using the following URI:

```
https://<host>:<port>/WickrIO/V1/Apps/<API Key>/GroupConvo
```

The body of this request will contain the information associated with the group conversation. The following is the format of the JSON data for the body of the request.

```
{
    "groupconvo": {
        "members" : [
            { "name" : "username001@wickr.com" },
            { "name" : "username002@wickr.com" }
        ]
    }
}
```

The response will either be an error with a description of that error or a successful response with the vGroupID of the newly created group conversation. The following is an example of a successful response:

```
{
    "vgroupid": "S0b503ae14cc896aad758ce48f63ac5fae0adccd78ef18cde82563c63b2c7761"
}
```

Get Group Conversations API

This API will return a list of group conversations that are known by the Wickr IO client. To get a list of group conversations send an HTTP GET to the following URI:

```
https://<host>:<port>/WickrIO/V1/Apps/<API Key>/GroupConvo
```

The Wickr IO client will respond with a JSON array of the group conversations. The format of the response will look like the following:

```
{
    "groupconvos": [
        {
            "members": [
                { "name" : "username001" },
                { "name" : "username002" }
            ],
            "ttl": "7776000",
            "bor": "0",
            "vgroupid":
 "S00bf0ca3169bb9e7c3eba13b767bd10fcc8f41a3e34e5c54dab8bflkjdfde"
        }
    ]
}
```

Get Group Conversation API

To get the details of a specific group conversation, send a HTTP GET command using the following URI with the vGroupID of the specific group conversation:

```
https://<host>:<port>/WickrIO/V1/Apps/<API Key>/GroupConvo/<vGroupID>
```

The Wickr IO client will respond with a JSON structure containing information for the specified conversation. The format of the response will look like the following:

```
{
    "rooms": [
        {
            "members": [
                { "name" : "username001" },
                { "name" : "username002" }
            ],
            "vgroupid":
 "S00bf0ca3169bb9e7c3eba13b767bd10fcc8f41a3e34e5c54dab8bflkjdfde"
        }
    ]
}
```

Delete Group Conversation API

In order to delete a group conversation, you will need to have the vGroupID associated with that conversation. You can use the get group conversations API to get the list of conversations known by the Wickr IO client, then determine which conversation to delete. Also, saving the vGroupID returned from the create group conversation API can be used as well.

To delete a group conversation, send an HTTP DELETE command using the following URI:

```
https://<host>:<port>/WickrIO/V1/Apps/<API Key>/GroupConvo/<vGroupID>
```

The group conversation with the same vGroupID will be deleted.

Sample integrations

This section describes how to set up and use the following public sample bot integrations that are available for Wickr IO:

- Wickr IO rekognition bot - Deploy a bot that allows storing, removal and retrieval of files for users on the same network.
- Wickr IO translation bot - Deploy a bot that facilitates language translation within Wickr conversations.
- Wickr IO Lex bot - Deploy an Amazon Lex chatbot within a web application to engage your web site visitors.

Wickr IO rekognition bot

Prerequisites

Before you start, be sure to complete the following before continuing with this guide:

- Follow steps 1 and 2 in *Quick start* to get ready to start the bot container.
- Set up AWS CLI, or an AWS credentials file with valid current credentials on your host machine. For additional information on how to get credentials, see: Authentication and access credentials for the AWS CLI
  - For this example, we will assume that the credentials on your host exist at `/home/ubuntu/credentials/rekognition/.aws`. This assumes that your username is ubuntu and that you have stored your credentials in the file `~/credentials/rekognition/.aws/credentials`.
  - For long term credentials be sure to follow best practices as outlined in Authentication and access credentials for the AWS CLI.

Deploy rekognition bot

Complete the following procedure to deploy a rekognition bot.

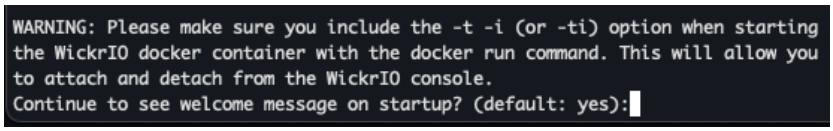
Step 1: Deploy and configure

1. Deploy and configure `wickrio-rekognition-bot` container on your host. For more information, see wickrio-rekognition-bot npm package.
2. Start the docker image on your host

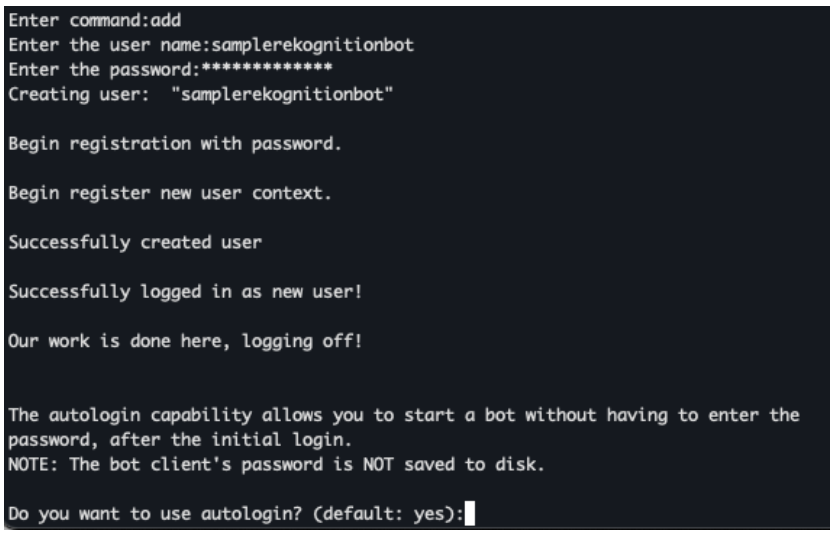
```
docker run -v ~/WickrIO:/opt/WickrIO -v /home/ubuntu/
        credentials/rekognition/.aws:/home/wickriouser/.aws -ti public.ecr.aws/
```

```
        x3s2s6k3/wickrio/bot-cloud:latest
```

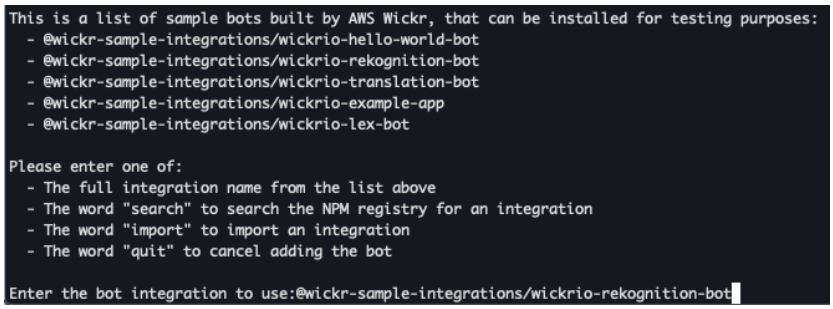
3. Select your preference for the welcome message.



4. At the **Enter command:** prompt, enter the command **add**.



5. Over the next several prompts, enter the username and password created in the previous steps.



6. You are prompted to select an integration. Type `@wickr-sample-integrations/wickrio-rekognition-bot`

7. Next, you are prompted for the AWS Region, and for the name of your profile in your credentials file that you set up earlier. If you are unsure of the AWS Region, you can use US East (N. Virginia). AWS Wickr is available in the following regions: [AWS Wickr Regional availability].

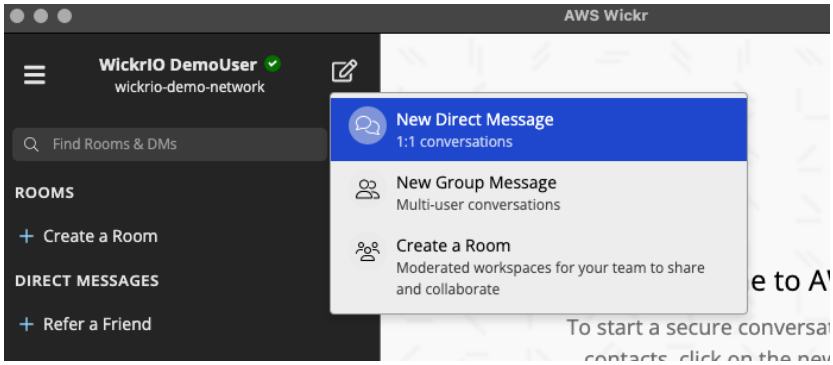
```
Copying @wickr-sample-integrations/wickrio-rekognition-bot from the NPM registry
Installing @wickr-sample-integrations/wickrio-rekognition-bot software
Begin configuration of @wickr-sample-integrations/wickrio-rekognition-bot software for samplerekognitionbot
Log level set to debug
Max file size set to 10m
Max number of files set to 5
Creating logger
Logger created
adminsOptional=false
adminsOptional=false
 Please enter the AWS region to use: (us-east-1) :us-east-1
 Please enter profile from your credentials file to use: (default) :default
Finished Configuring WPM!
Finished Configuring pid location file!
Finished Configuring!

Integration files written to:
/opt/WickrIO/clients/samplerekognitionbot/integration/@wickr-sample-integrations/wickrio-rekognition-bot

End of setup of @wickr-sample-integrations/wickrio-rekognition-bot software for samplerekognitionbot
**********************************************************************
Successfully added record to the database!
```

Step 2: Start the bot

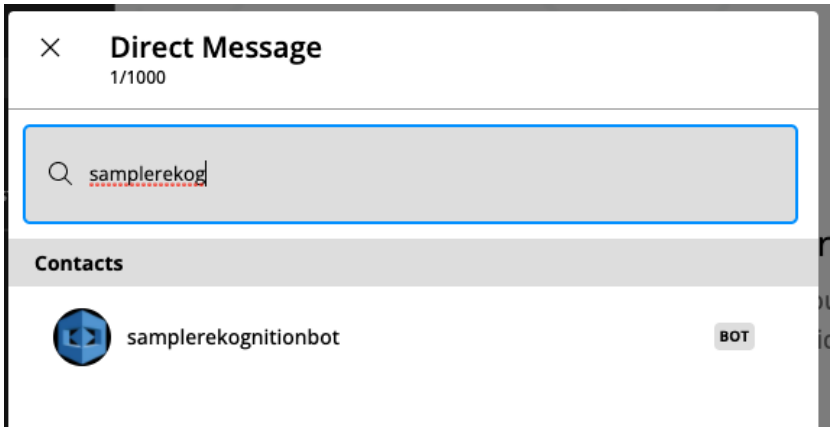
1. Use the **list** command to view a list of available bots.
2. Using the number of the bot that you just created (0 in the example), type the command **start #**, where # is the bot number (0 in the example).
3. Enter the password for the bot.
4. Use the **list** command to verify that the bot is running.

Step 3: Interact with the bot

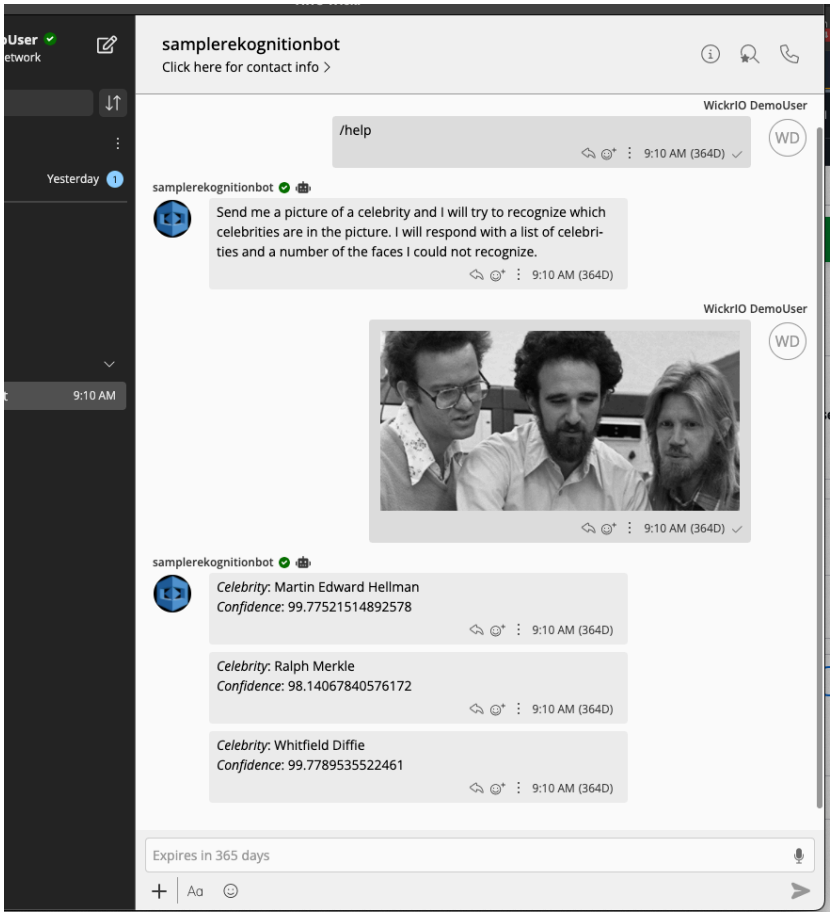
1. In the top right corner of the navigation panel, select the new message button.



2. In the pop up menu, choose **New Direct Message**.
3. Search for your bot by display name.



4. Select your bot for a direct message, and send a message.



Wickr IO translation bot

Prerequisites

Before you start, be sure to complete the following before continuing with this guide:

- Follow steps 1 and 2 in *Quick start* to get ready to start the bot container.
- Set up AWS CLI, or an AWS credentials file with valid current credentials on your host machine. For additional information on how to get credentials, see: Authentication and access credentials for the AWS CLI
  - For this example, we will assume that the credentials on your host exist at `/home/ubuntu/credentials/translation/.aws`. This assumes that your username is ubuntu and that you have stored your credentials in the file `~/credentials/translation/.aws/credentials`.
  - For long term credentials be sure to follow best practices as outlined in Authentication and access credentials for the AWS CLI.

Deploy translation bot

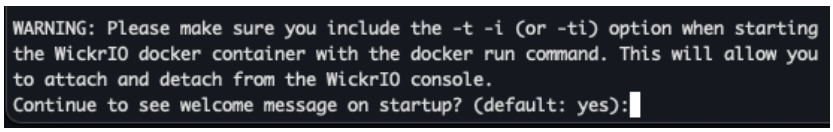
Complete the following procedure to deploy a translation bot.

Step 1: Deploy and configure

1. Deploy and configure `wickrio-translation-bot` container on your host. For more information, see wickrio-translation-bot .
2. Start the docker image on your host

```
docker run -v ~/WickrIO:/opt/WickrIO -v /home/ubuntu/
        credentials/translation/.aws:/home/wickriouser/.aws -ti public.ecr.aws/
        x3s2s6k3/wickrio/bot-cloud:latest
```

3. Select your preference for the welcome message.



4. At the **Enter command:** prompt, enter the command **add**.

```
Enter command:add
Enter the user name:sampletranslationbot
Enter the password:*************
Creating user:  "sampletranslationbot"

Begin registration with password.

Begin register new user context.

Successfully created user

Successfully logged in as new user!

Our work is done here, logging off!


The autologin capability allows you to start a bot without having to enter the
password, after the initial login.
NOTE: The bot client's password is NOT saved to disk.

Do you want to use autologin? (default: yes):
```

5. Over the next several prompts, enter the username and password created in the previous steps.

```
This is a list of generally available bots that are built and maintained by AWS Wickr:
  - wickrio-broadcast-bot
  - wickrio_web_interface

This is a list of sample bots built by AWS Wickr, that can be installed for testing purposes:
  - @wickr-sample-integrations/wickrio-hello-world-bot
  - @wickr-sample-integrations/wickrio-rekognition-bot
  - @wickr-sample-integrations/wickrio-translation-bot
  - @wickr-sample-integrations/wickrio-example-app
  - @wickr-sample-integrations/wickrio-lex-bot

Please enter one of:
  - The full integration name from the list above
  - The word "search" to search the NPM registry for an integration
  - The word "import" to import an integration
  - The word "quit" to cancel adding the bot

Enter the bot integration to use:@wickr-sample-integrations/wickrio-translation-bot
****************************************************************
```

6. You are prompted to select an integration. Type `@wickr-sample-integrations/wickrio-translation-bot`
7. Next, you are prompted for the AWS Region, and for the name of your profile in your credentials file that you set up earlier. If you are unsure of the AWS Region, you can use US East (N. Virginia)(us-east-1). AWS Wickr is available in the following regions: AWS Wickr Regional availability.

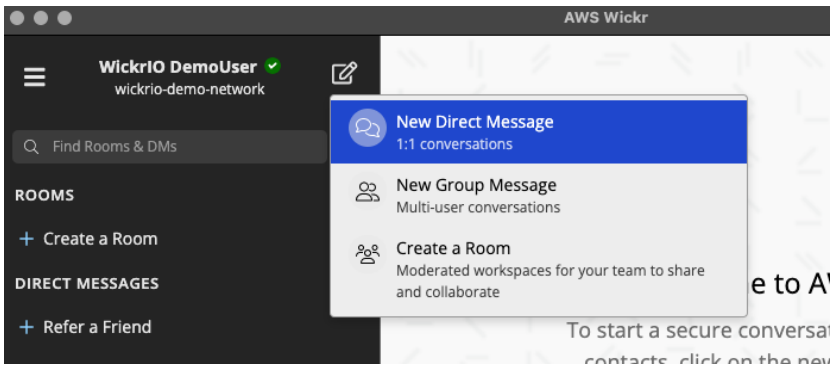
```
Begin setup of @wickr-sample-integrations/wickrio-translation-bot software for sampletranslationbot
v20.17.0
Copying @wickr-sample-integrations/wickrio-translation-bot from the NPM registry
Installing @wickr-sample-integrations/wickrio-translation-bot software
Begin configuration of @wickr-sample-integrations/wickrio-translation-bot software for sampletranslationbot
Log level set to debug
Max file size set to 10m
Max number of files set to 5
Creating logger
Logger created
adminsOptional=false
adminsOptional=false
 Please enter the AWS region to use: (us-east-1) :us-east-1
 Please enter profile from your credentials file to use: (default) :bot
Finished Configuring WPM!
Finished Configuring pid location file!
Finished Configuring!

Integration files written to:
/opt/WickrIO/clients/sampletranslationbot/integration/@wickr-sample-integrations/wickrio-translation-bot

End of setup of @wickr-sample-integrations/wickrio-translation-bot software for sampletranslationbot
**********************************************************************
Successfully added record to the database!
```

Step 2: Start the bot

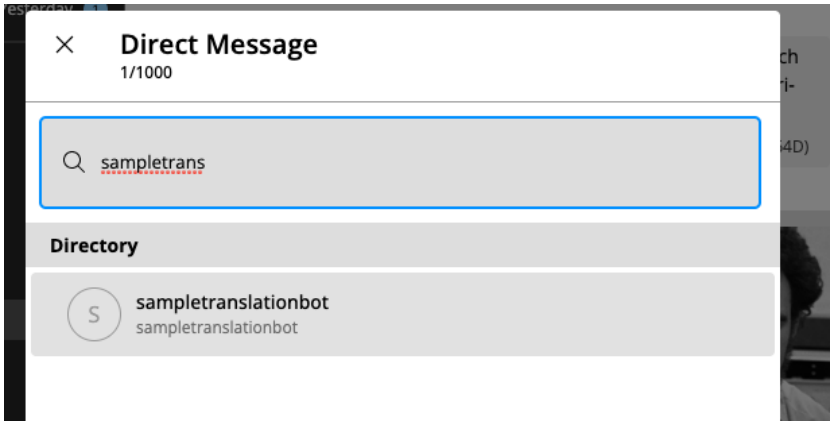
1. Use the **list** command to view a list of available bots.
2. Using the number of the bot that you just created (0 in the example), type the command **start #**, where # is the bot number (0 in the example).
3. Enter the password for the bot.
4. Use the **list** command to verify that the bot is running.

Step 3: Interact with the bot

1. In the top right corner of the navigation panel, select the new message button.



2. In the pop up menu, choose **New Direct Message**.
3. Search for your bot by display name.



4. Select your bot for a direct message, and send a message.

   The example below illustrates how the bot translates between English to French.

Wickr IO lex bot

Prerequisites

Before you start, be sure to complete the following before continuing with this guide:

- Follow steps 1 and 2 in *Quick start* to get ready to start the bot container.
- Set up AWS CLI, or an AWS credentials file with valid current credentials on your host machine. For additional information on how to get credentials, see: Authentication and access credentials for the AWS CLI

- For this example, we will assume that the credentials on your host exist at `/home/ubuntu/credentials/lex/.aws`. This assumes that your username is ubuntu and that you have stored your credentials in the file `~/credentials/lex/.aws/credentials`.
- For long term credentials be sure to follow best practices as outlined in Authentication and access credentials for the AWS CLI.
- Make sure to collect the following pieces of information
  - Bot ID
  - Deployment Alias ID

Deploy lex bot

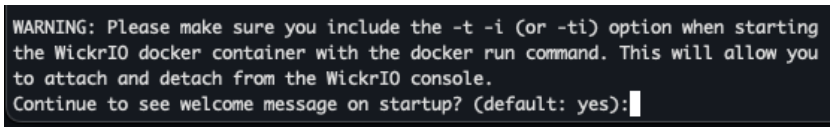
Complete the following procedure to deploy a lex bot.

Step 1: Deploy and configure

1. Deploy and configure `wickrio-rekognition-bot` container on your host. For more information, see wickrio-lex-bot .
2. Start the docker image on your host

```
docker run -v ~/WickrIO:/opt/WickrIO -v /home/ubuntu/
        credentials/lex/.aws:/home/wickriouser/.aws -ti public.ecr.aws/
        x3s2s6k3/wickrio/bot-cloud:latest
```

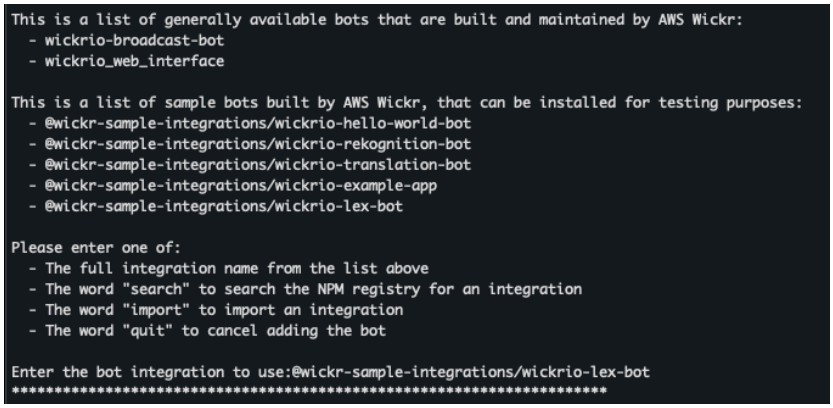
3. Select your preference for the welcome message.



4. At the **Enter command:** prompt, enter the command **add**.

```
Enter command:add
Enter the user name:samplelexbot
Enter the password:************
Creating user:  "samplelexbot"

Begin registration with password.

Begin register new user context.

Begin register existing user context.

Successfully created user

Successfully logged in as new user!

Our work is done here, logging off!
```

5. Over the next several prompts, enter the username and password created in the previous steps.



6. You are prompted to select an integration. Type `@wickr-sample-integrations/wickrio-lex-bot`
7. Next, you are prompted for the AWS Region, and for the name of your profile in your credentials file that you set up earlier. If you are unsure of the AWS Region, you can use US East (N. Virginia). AWS Wickr is available in the following regions: [AWS Wickr Regional availability](#).

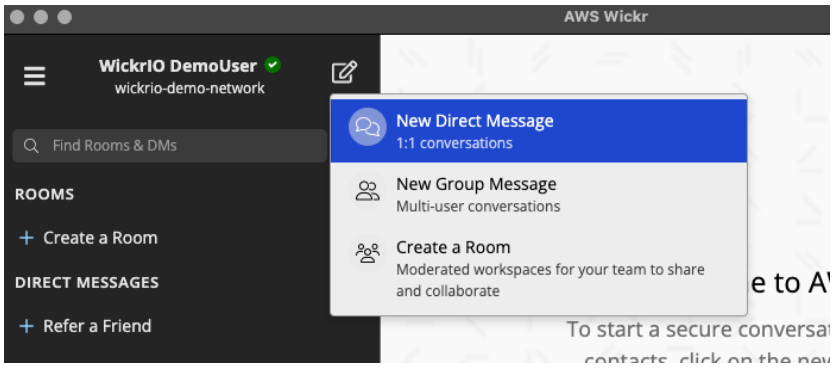
```
Copying @wickr-sample-integrations/wickrio-lex-bot from the NPM registry
Installing @wickr-sample-integrations/wickrio-lex-bot software
Begin configuration of @wickr-sample-integrations/wickrio-lex-bot software for samplelexbot
Log level set to debug
Max file size set to 10m
Max number of files set to 5
Creating logger
Logger created
adminsOptional=false
adminsOptional=false
 Please enter the AWS region to use: (us-east-1) :us-east-1
 Please enter profile from your credentials file to use: (default) :bot
 Please enter Lex bot ID: :TQMIDUPIJA
 Please enter Lex alias ID: :TSTALIASID
Finished Configuring WPM!
Finished Configuring pid location file!
Finished Configuring!

Integration files written to:
/opt/WickrIO/clients/samplelexbot/integration/@wickr-sample-integrations/wickrio-lex-bot

End of setup of @wickr-sample-integrations/wickrio-lex-bot software for samplelexbot
*********************************************************************
Successfully added record to the database!
```

Step 2: Start the bot

1. Use the **list** command to view a list of available bots.
2. Using the number of the bot that you just created (0 in the example), type the command **start #**, where # is the bot number (0 in the example).
3. Enter the password for the bot.
4. Use the **list** command to verify that the bot is running.

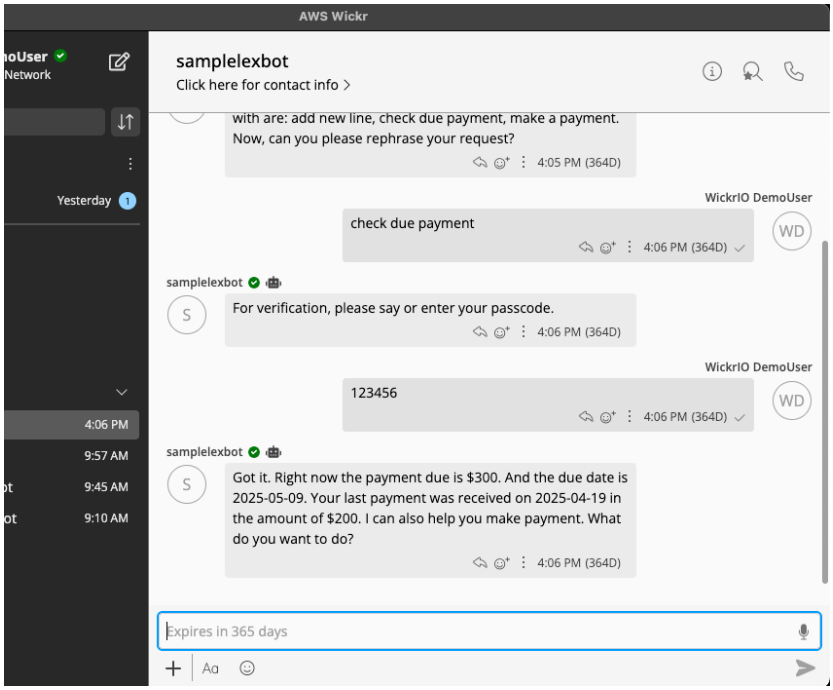
Step 3: Interact with the bot

1. In the top right corner of the navigation panel, select the new message button.



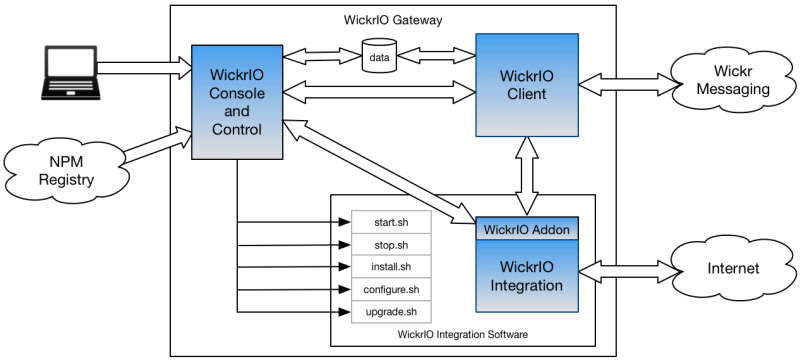
2. In the pop up menu, choose **New Direct Message**.
3. Search for your bot by display name.

4. Select your bot for a direct message, and send a message.



Develop a custom Wickr IO integration on AWS Wickr

High Level Overview



To customize your experience with integrations in AWS Wickr, Wickr IO offers a JavaScript library which makes it easy to develop your own bots. This document contains the process of creating a new integration, an “emoji bot,” which responds to messages with a random emoji.

Integration setup

Complete the following procedure to setup a new custom Wickr IO integration.

1. Create a new directory and install the Wickr IO bot API with NPM.

```
mkdir wickrio-emoji-bot
cd wickrio-emoji-bot
npm install wickrio-bot-api
```

2. In the root directory of your new package, create a file named `processes.json` with the following contents:

```
{
  "apps": [
    {
      "name": "wickrio-emoji-bot",
      "env": {
        "tokens": {}
      }
    }
```

```
  ]
}
```

This file is used by the Wickr IO bot API library to provide custom configuration for your integration. Values provided in the `env.tokens` property will be set as environment variables in your integration.

3. In addition to the `processes.json` file, you need to create the following two scripts in order for your bot to start up successfully:

   1. **install.sh**

      This file is required when creating the bot. There are no special install time requirements, you can exit with a successful status (0).

      ```
      #!/bin/sh
      exit 0
      ```

   2. **start.sh**

      This script is used by the Wickr IO integration to start your bot. The code that is created below will be in the file `index.js`. Configure the startup script to run this file.

      ```
      #!/bin/bash
      node index.js "$@"
      ```

4. Make these scripts executable by running `chmod`:

   ```
   chmod +x {install,start}.sh
   ```

5. Next, create an `index.js` file which serves as the main entrypoint for our bot:

   ```
   // index.js
   ```

```
const WickrIOBotAPI = require('wickrio-bot-api')

const bot = new WickrIOBotAPI.WickrIOBot()
const WickrIOAPI = bot.apiService().WickrIOAPI

async function listen() {}

async function main() {
  const username = process.argv[2]
  if (!username) throw new Error('Missing username')

  const status = await bot.start(username)
  if (!status) throw new Error('Unable to start bot')

  bot.startListening(listen);
}

main()
```

Add a custom slash command

To add custom behavior, you have to make changes to `index.js`.

Complete the following procedure to add a custom slash command.

1. Add a `getEmoji` function. This will return a random emoji.
2. Modify the `listen` function. This will allow it to respond to the slash-command /emoji with a random emoji from the list.

```
// index.js
const WickrIOBotAPI = require('wickrio-bot-api')

const bot = new WickrIOBotAPI.WickrIOBot()
const WickrIOAPI = bot.getWickrIOAddon()

function getEmoji() {
  const emojis = [
    "#", "#", "#", "#", "#", "#", "#", "#", "#", "#", "#",
    "#", "#", "#", "#", "#", "#", "#", "#", "#"
  ]
```

```
  return emojis[Math.floor(Math.random() * emojis.length)]
}

async function listen(input) {
  const msg = bot.parseMessage(input)
  if (!msg) return

  switch (msg.command) {
    case "/emoji":
      await WickrIOAPI.cmdSendRoomMessage(msg.vgroupid, getEmoji())
      break
  }
}

async function main() {
  const username = process.argv[2]
  if (!username) throw new Error('Missing username')

  const status = await bot.start(username)
  if (!status) throw new Error('Unable to start bot')

  bot.startListening(listen)
}

main()
```

Build

Custom integrations in Wickr IO must be made available to the Wickr IO Docker container as a tarball named `software.tar.gz`. You can build that tarball from your source directory:

```
tar czf software.tar.gz –exclude=software.tar.gz .
```

Here's an example of what the output to this command will look like:

```
tar czf software.tar.gz –exclude=software.tar.gz .
```

```
tar: .: file changed as we read it
```

After you run this command, you will have the `software.tar.gz` file containing your source code and dependencies for your bot. Upload it to the server where you want to deploy your bot integration to continue on with the next deployment step.

Deploy

In this section you can deploy your custom integration using the Wickr IO Docker container. These commands should be run on a host which has Docker installed, access to the internet, and has the `software.tar.gz` archive for your custom integration on its file system.

If you haven't already created your bot user in the AWS Wickr console, do that now. You'll need the bot username and password in order to register your bot user for your new custom integration.

Create a bot data directory

Complete the following procedure to create a bot data directory.

1. On your server, create a new directory for all of the files which will be associated with your bot. In this directory, you will place the `software.tar.gz` file, a client configuration file, and a directory to hold all of the data for the Wickr IO container.

   ```
   mkdir -p emoji-bot/data
   cd emoji-bot
   cp ../path/to/software.tar.gz . # Update this path to the correct location
   ```

2. Create a file named `clientConfig.json` which contains the information needed to start your bot. Replace `BOT_USERNAME` and `BOT_PASSWORD` with the credentials for your bot.

   ```
   {
     "clients": [
       {
         "integration": "wickrio-emoji-bot",
         "name": "BOT_USERNAME",
         "password": "BOT_PASSWORD",
   ```

```
      "tokens": []
    }
  ]
}
```

Your directory should have the following structure:

```
.
### clientConfig.json  # Your bot client's configuration file
### data               # The data directory for the WickrIO container
### software.tar.gz    # Your custom integration code
```

Start the container

You can now start the Wickr IO container to bring your new custom integration online. The directory which the `software.tar.gz` file is mounted to must match the integration named supplied in your `clientConfig.json` file.

In the example, the integration is named `wickrio-emoji-bot`. The tarball must be mounted to `/usr/lib/wickr/integrations/software/wickrio-emoji-bot/` .

```
docker run -d --restart=always \
    -v ./data:/opt/WickrIO \
    -v ./clientConfig.json:/usr/local/wickr/WickrIO/clientConfig.json:ro \
    -v ./software.tar.gz:/usr/lib/wickr/integrations/software/wickrio-emoji-bot/
software.tar.gz:ro \
    public.ecr.aws/x3s2s6k3/wickrio/bot-cloud:latest
```

You can now interact with your new integration by sending it the `/emoji` command in your AWS Wickr client.

Node.js Addon API

This section describes the Wickr IO Node.js addon and how to use it with several examples. The APIs provided by the Wickr IO Node.js addon are low-level APIs, using the APIs provided by the

Wickr IO Bot API provide a higher-level approach to some of the main aspects of an Wickr IO integration interfacing with a Wickr IO client. You will still need to use the Wickr IO Node.js addon to perform most interactions with the Wickr IO client. The APIs supported by the Wickr IO Node.js addon allow you to access all of the necessary functionality from the Wickr IO client to communicate with other Wickr users within the Wickr network.

The Wickr IO Node.js addon is published to the default NPM registry. The name of the published module is `wickrio_addon`. It supports a specific set of functions consistent with the Wickr IO REST API.

When using the Wickr IO Node.js addon you will have access to one Wickr IO client at a time. It is not currently possible for an integration to communicate with more than one Wickr IO client at the same time via the addon. Interaction with the Node.js is as follows:

1. Initialize the Wickr IO Node.js addon interface. This is done by calling the clientInit() API, and supplying the user name of the Wickr IO client that is going to be used. The start() API provided by the Wickr IO Bot API uses the clientInit() API to initiate a connection to the Wickr IO client. We recommend using that API instead.
2. Interact with the Wickr IO client by calling the appropriate APIs.
3. When your program is complete then call the closeClient() API to gracefully stop processing. The close() API, provided by the Wickr IO Bot API, uses the closeClient() API to shut down the connection to the Wickr IO client.

The addon APIs are described in detail in the following sections.

Startup and Shutdown APIs

This section describes the APIs used to start and stop the connection between the Wickr IO integration and the Wickr IO client.

The interface between the Wickr IO integration and the Wickr IO client needs to be initialized. Initialization includes identifying the Wickr IO client and making sure the Wickr IO client is in the appropriate state, for example running and logged into the Wickr network. Once the interface between the client and the integration has been initialized you can start using the other Wickr IO Node.js addon APIs. Also, the interface should be gracefully shutdown when you are done using the APIs.

> **Note**
>
> Using the Wickr IO Bot API provides a higher-level API (start()) that will call the appropriate APIs in this addon.

clientInit(string *clientName*)

Before accessing any of the Wickr IO Node.js addon APIs you will need to run the "clientInit(*clientName*)". The only argument is the user name associated with the Wickr IO client.

> **Note**
>
> If you use the Wickr IO Bot API, the call to the start() API will call the clientInit() API.

closeClient()

This API will close the currently open client object(s). This should be called when done interacting with the client set in the "clientInit()" API.

> **Note**
>
> If you use the Wickr IO Bot API, the call to the close() API will call the closeClient() API.

isConnected(int *seconds*)

The isConnect() API checks if there is a valid connection from the calling Wickr IO integration to the Wickr IO client. The call will wait the input number of seconds for a connection. The API returns true if a response was received from the client within the amount of seconds input, otherwise it will return false.

> **Warning**
>
> If true is returned it does not mean the client is prepared to handle other requests yet. Use the `getClientState()` API to make sure the client is in the appropriate state. For most APIs the client should be in the RUNNING state.

> **Note**
>
> The Wickr IO Bot API's start() API uses this API to make sure the connection to the client exists.

getClientState()

This API retrieves the current state of the Wickr IO client the integration is connected with. The value returned is one of the following possible values:

- LOGGINGIN
- NOTRUNNINNG
- RUNNING
- SHUTTINGDOWN
- STARTING
- UNINITIALIZED

It is important to make sure the Wickr IO client is in the RUNNING state before performing any API calls that require access to the Wickr network. Doing so before then may have unpredictable results.

> **Note**
>
> The Wickr IO Bot API's start() API uses this API repetitively to make sure the client is in the RUNNING state before returning.

Configuration API

The configuration API is used by the integration to configure specific Wickr IO client modes of operation. Since the Wickr IO client is a separate software module this API will provide a method to modify how the client operates. Typically, the use of these APIs is done after the interface to the Wickr IO client has been initialized.

cmdSetControl(string *configKey*, string *configValue*)

This API will tell the Wickr IO client to set the specific *configKey* to the input *configValue*. You will have to make a call to cmdSetControl() for each *configKey* value that is being set. The following is a list of *configKey* values that can be set:

Config Key	Description
attachLifeMinutes	This is a number that represents the number of minutes an attachmen t (file) will remain on the Wickr IO client. After that amount of time the attachment will be removed. A value of 0 will keep attachments on the system indefinitely, this is the default value. It is highly recommended that this value is set to something other than 0.
doreceive	If set to 'false' the Wickr IO client will NOT forward incoming Wickr messages to the integration software. This is useful for transmit only integrations. The default value is 'true'.
duration	This value is used to make the Wickr IO client and integration to perform a restart after a number of seconds. A value of 0 will not perform a restart. This value is helpful for testing, or if you wish to perform periodic restarts. The default value is '0'.

Statistics APIs

APIs are provided to retrieve messaging statistics that are maintained by the Wickr IO client. You can get and clear the retrieved statistics.

cmdGetStatistics()

This API will return a list of messaging and error statistics, for example.

```
{
    "statistics": {
        "message_count": 5,
        "pending_messages": 0,
        "sent": 7,
        "received": 3,
        "sent_errors": 1,
        "recv_errors": 1
```

```
    }
}
```

The following table describes each of the statistics that can be returned by this API:

Statistics	Description
message_count	The number of received messages that are currently queued to a conversation on the Wickr IO client. These queues feed into the main receive queue that is used by the message callback and retrieval APIs.
outbox_sync	The number of outbox sync messages received. Outbox sync messages are messages that were sent by another device for this Wickr IO client. This is only valid when there are multiple devices configured for a Wickr IO client, which is not typical.
pending_callback_messages	The number of messages on the callback message queue. These are messages received by the Wickr IO client, that are waiting to be sent to a callback process. This number will be decremented for each message that is retrieved from the Wickr IO client using the cmdGetReceivedMessage() API or received by the integration software successfully via the asynchronous message handling (i.e. callback).
pending_messages	The number of messages that are currently queued to be sent from the Wickr IO client to Wickr clients on the Wickr network.
received	The number of messages that the Wickr IO client has received.
recv_errors	The number of errors that occurred while receiving messages.
sent	The number of messages that have been sent by the Wickr IO client.
sent_errors	The number of errors that have occurred while trying to send messages.

cmdClearStatistics()

This API will clear the current statistics that are saved in the client.

Wickr Client APIs

This section describes APIs that provide information about Wickr Clients. You can use these APIs to get information about a Wickr client that the bot can communicate with.

cmdGetUserInfo(string users[])

This API returns information about the each of the users from the input array of Wickr user IDs. The value returned is a JSON object with two arrays, one for the Wickr users that exist and one for the Wickr users that do not exist. The following is an example of the JSON returned:

```
{
    "users" : [
        { "name" : "bobsmith@company.com", "full_name" : "Bob Smith", "is_bot",
 false" },
        { "name" : "joebrown@companuy.com", "full_name" : "Joe Brown", "is_bot",
 false" }
        ],
    "failed" : [ "failedUser1", "failedUser2" ]
}
```

> **Warning**
>
> This API requires interaction with the Wickr server, so keep the number of users on the input list small so the response does not take too long.

cmdGetClientInfo()

This API will retrieve the bot client information. The values are returned in a JSON list of objects.

```
{
    "version" : "<bot client's version>",
    "organization" : "Wickr, Inc."
}
```

This API can be used to verify the version of the client.

cmdSetVerificationMode(string mode)

This API is used to set the verification mode that the bot client will operate under. The possible values for the input mode are "automatic" or "manual", all other values will return an error. If the verification mode is set to "automatic" then any client the bot client interacts with becomes unverified, the bot client will automatically put that client into the verified state. If the verification mode is set to "manual" then it is up to the bot integration to peridically get the verification list, using the "cmdGetVerificationList" API and then calling the "cmdVerifyUsers" or "cmdVerifyAllUsers" APIs.

cmdGetVerificationList(string mode)

This API will return a list of users that have become unverified. The mode argument is optional. Normally, this API will return users that have a failed verification status. When you call this function with a mode value of "all" then this API will return users that are not verified, meaning they can have a verification status of failed, unverified or pending. The value returned will be a JSON string with a list of users and their verification status. These verification functions are necessary when verification is done manually but the bot integration.

The following is a sample response:

```
{
    "users" : [
        { "user": "wickrID1", "reason": "failed" },
        { "user": "wickrID2", "reason": "unverified" },
        { "user": "wickrID3", "reason": "pending" }
    ]
}
```

cmdVerifyUsers(string users[])

This API will verify all of the users in the input users array. The verification status for each of these users will be changed to verified. You will only need to use this API if your bot was setup to do verification manually.

cmdVerifyAll()

This API will verify all users that are in the unverified or failed verification state. This will only be necessary if the bot is doing verification manually.

Secure Room Conversation APIs

This section describes the APIs that perform operations on Wickr secure rooms. The operations you can perform include adding, modifying, deleting and retrieving secure rooms. For APIs where you are dealing with a specific secure room, you will need to have the *VGroupID* that is associated with that room.

cmdAddRoom(string members[], string moderators[], string title, string desc, string ttl, string bor)

This API creates a new secure room. The arguments of this request will contain the information associated with the room.

The members and moderators arguments are arrays of strings, that are Wickr IDs. The members array is the complete list of members of the secure room. The moderators array is a list of the moderators of the secure room. The moderators must also be in the members list. There must be at least one moderator in the room.

The ttl is the time to live value. The bor is the burn on read value. These values are strings but the contents of the string is a number. The ttl and bor values are optional. If the bor value is included then the ttl value must also be included.

```
{
    "vgroupid": "S0b503ae14cc896aad758ce48f63ac5fae0adccd78ef18cde82563c63b2c7761"
}
```

The response will either be an error with a description of that error or a successful response with the vGroupId of the newly created room.

cmdDeleteRoom(string vgroupid)

In order to delete a secure room, you will need to have the vGroupID associated with that room. You can use the get rooms API to get the list of rooms known by the Wickr IO client, then determine which room to delete. Also, saving the vGroupID returned from the add room API can be used as well.

cmdGetRoom(string *vgroupid*)

This API will return details of a specific secure room conversation. The Wickr IO client will respond with a JSON structure containing information for the specified conversation.

```
{
    "rooms": [
        {
            "description": "Room description",
            "masters": [
                { "name" : "username001" }
            ],
            "members": [
                { "name" : "username001" },
                { "name" : "username002" }
            ],
            "title": "Room Title",
            "ttl": "-1",
            "vgroupid":
 "S00bf0ca3169bb9e7c3eba13b767bd10fcc8f41a3e34e5c54dab8bflkjdfde"
        }
    ]
}
```

cmdGetRooms()

This API will return a list of secure rooms that are known by the Wickr IO client. The Wickr IO client will respond with a JSON array of the secure rooms that the Wickr IO client is a member of. The following is an example of what the JSON returned from this API:

```
{
    "rooms": [
        {
            "description": "Room description",
            "masters": [
                { "name" : "username001" }
            ],
            "members": [
                { "name" : "username001" },
                { "name" : "username002" }
            ],
            "title": "Room Title",
```

```
            "ttl": "7776000",
            "bor": "0",
            "vgroupid":
 "S00bf0ca3169bb9e7c3eba13b767bd10fcc8f41a3e34e5c54dab8bflkjdfde"
        }
    ]
}
```

cmdLeaveRoom(string vgroupid)

This API will instruct the Wickr IO client to leave the secure room identified by the input vGroupID. In order to leave a secure room, you will need to have the vGroupID associated with that room. You can use the get rooms API to get the list of rooms known by the Wickr IO client, then determine which room to leave. Also, saving the vGroupID returned from the create room API can be used as well.

cmdModifyRoom(string vgroupid, string members[], string moderators[], string title, string description, string ttl, string bor)

This API is used to modify some of the settings associated with a room. The following room attributes can be modified using this API:

- TTL
- BOR
- Description
- Title
- Members
- Moderators

The Wickr IO client must be a moderator for the room identified by the input vGroupID, otherwise the request will fail.

Group Conversation APIs

This section describes the APIs associated with group conversations. Using these APIs, you can create, get or delete group conversations that the Wickr IO client is a part of.

cmdAddGroupConvo(string members[], string ttl, string bor)

This API will create a new group conversation. The members argument is required, and the ttl and bor values are optional. The response will either be an error with a description of that error or a successful response with the vGroupID of the newly created group conversation.

```
{
    "vgroupid": "G0b503ae14cc896aad758ce48f63ac5fae0adccd78ef18cde82563c63b2c7761"
}
```

cmdDeleteGroupConvo(string vgroupid)

This API will instruct the Wickr IO client to leave a group conversation. You can only actually delete a group conversation if the client is the last member of the group conversation.

In order to delete a group conversation, you will need to have the vGroupID associated with that conversation. You can use the get group conversations API to get the list of conversations known by the Wickr IO client, then determine which conversation to delete. Also, saving the vGroupID returned from the create group conversation API can be used as well. The group conversation with the same vGroupID will be deleted.

cmdGetGroupConvos()

This API will return a list of group conversations that are known by the Wickr IO client. The Wickr IO client will respond with a JSON array of the group conversations.

```
{
    "groupconvos": [
        {
            "members": [
                { "name" : "username001" },
                { "name" : "username002" }
            ],
            "ttl": "7776000",
            "bor": "0",
            "vgroupid":
 "G00bf0ca3169bb9e7c3eba13b767bd10fcc8f41a3e34e5c54dab8bflkjdfde"
        }
    ]
}
```

cmdGetGroupConvo(string vgroupid)

This API will return details of a specific group conversation. The Wickr IO will respond with a JSON structure containing information for the specified conversation.

```
{
    "rooms": [
        {
            "members": [
                { "name" : "username001" },
                { "name" : "username002" }
            ],
            "vgroupid":
 "G00bf0ca3169bb9e7c3eba13b767bd10fcc8f41a3e34e5c54dab8bflkjdfde"
        }
    ]
}
```

Receive Message APIs

These messaging APIs are used to retrieve Wickr messages received by the Wickr IO client. Messages received by the Wickr IO client can be retrieved using one of the following methods:

- Explicitly polling for the received messages
- Asynchronously receiving messages
- Setting a URL where received messages will be posted to.

The Wickr IO APIs will only support one receive message method at a time. When a message has successfully been transferred from the Wickr IO client to the integration software that message will be removed from the Wickr IO client queue. The Wickr IO client will not save messages after completion of the transfer.

The format of the messages received is described in the message format section.

cmdGetReceivedMessage()

This API will retrieve the next message waiting to be read. Each call to this API will return just one message if any are waiting to be read. After the message is retrieved it will be removed from the Wickr IO client database.

cmdStartAsyncRecvMessages(function callback(string))

This API will initiate the asynchronous reception of received messages. The input callback argument will be called when a message is received by the associated client.

Any implementation that uses this API must make sure events can be handled so that the message callback may be called.

cmdStopAsyncRecvMessages()

This API will stop the asynchronous reception of received messages.

cmdSetMsgCallback(string url)

Use this API to set a URL that will be used by the client to send received messages to. Any messages received after this API is performed will be sent to the defined URL. When using this method of receiving messages be careful to make sure the software running on the Wickr IO Docker image can access the URL.

cmdGetMsgCallback()

Use this API to get the currently set message callback URL. If there is a URL callback defined the following is the format of the body.

```
{
    "callback": "https://localhost:4008"
}
```

cmdDeleteMsgCallback()

If the URL callback is no longer needed or you need to switch to receive messages asynchronously then delete the existing message URL callback. This API will delete the current message callback.

Transmit Message Arguments

There are several arguments to the transmit APIs that need more detailed descriptions.

Message Meta Arguments

New to the 5.81 version of WickrIO, is the support for button and table GUI widgets. These GUI widgets are only supported in text messages, file messages do not support them. The transmit APIs have been modified to support an optional argument (messagemeta) that identifies these buttons or tables. The messagemeta argument is a JSON string that identifies the GUI elements associated with the message being transmitted. The following figure shows the JSON associated with two buttons, one is a normal message button and the other is a location button:

```
{
    "buttons" : [
        {
            "type": "message",
            "text": "Button Text",
            "message": "/action"
        },
        {
            "type": "getlocation",
            "text": "Send Location"
        }
    ]
}
```

The "type" object identifies the type of the button, and the "text" identifies the text that will be displayed on the button to the user. The "message" object of the message button is what the client will send back to the bot when the button is selected. The "getlocation" type of button, when selected the client will send the client's location to the bot.

The table GUI widget is used to display a selectable list of information on the client. The messagemeta JSON string can contain a "table" object and a "textcut" object. The "textcut" object is optional. The "table" object contains all the details of the table to be shown. The "textcut" object contains a list of values that indicate which characters in the message text should be cut if the client supports the table GUI widget. Normally, if a client does not support the table GUI widgets it will just display the message text. If the client does support the table GUI widgets it will display the message text, minus the text referenced by the "textcut" values, and then the list GUI. The following is a sample.

```
{
    "table" : {
        "name": "Table heading",
        "firstcolname": "Column 1",
        "secondcolname": "Column 2",
        "actioncolname": "Action",
        "rows": [
            {
                "firstcolvalue": "123",
                "secondcolvalue": "Hello",
                "response": "1"
            },
            {
                "firstcolvalue": "2838",
                "secondcolvalue": "There",
                "response": "2"
            }
        ]
    },
    "textcut" : [
        { "startindex": 0, "endindex": 75 }
    ]
}
```

The sample above shows all of the possible objects associated with the "table" and "textcut" objects. The "textcut" array is optional. The "secondcolname" and "secondcolvalue" objects are optional. The table can have one or two columns.

To include buttons or lists in your text messages you will create a JSON string and that wil be the messagemeta argument to the message sending APIs (shown below). The following is an example of the Javascript code for creating the messagemeta string for some buttons. When the button is selected the 'message' value will be sent to the bot.

```
const messagemeta = {
    buttons: [
        {
            type: 'message',
            text: 'yes',
            message: 'yes',
        },
        {
            type: 'message',
```

```
            text: 'no',
            message: 'no',
        }
    ],
}
const messagemetastring = JSON.stringify(messagemeta)
```

The following is an example Javascript for creating the message meta string for a list. The action column contains the value that will be returned to the bot when that item is selected. In this case it will be the number '1', '2' or '3'.

```
const users = [ 'user1@somewhere.com', 'user2@somewhere.com', 'user3@somewhere.com' ]

let messagemeta = {
    table: {
        name: 'List of Users',
        firstcolname: 'User',
        actioncolname: 'Select',
        rows: [],
    },
    textcut: [
        {
            startindex: 0,
            endindex: entriesString.length - 1,
        },
    ],
}

for (let i = 0; i < users.length; i++) {
    const response = i + 1
    const row = {
        firstcolvalue: users[i],
        response: response.toString(),
    }
    messagemeta.table.rows.push(row)
}

const messagemetastring = JSON.stringify(messagemeta)
```

Message ID Arguments

The messageID argument for the transmit functions is used to track the status of transmits. Bot integrations like the broadcast bot use the messageID values to track the process of a broadcast.

The messageID can be used later to retrieve the status of the transmission of all messages associated with that messageID value.

Flags Arguments

The flags arguments to the transmit functions is not fully defined. It will be defined in a future document.

Transmit Message APIs

The transmit message APIs support transmitting normal messages as well as files. Messages and files will be transmitted to specific 1-on-1, secure room, or group conversations on the Wickr network via the Wickr IO client. For secure rooms and group conversations you will need to have the vGroupID associated with the specific conversation.

Some of the arguments to these functions are optional. Required arguments will always be listed first in the function definitions. The order of the arguments must follow the defined function signature. If you are going to use an optional argument that is after one you are not going to use then you will have to pass an appropriate value for the optional argument you are not using (i.e. "" for string arguments, [] for array arguments).

cmdSend1to1Message(string users[], string message, string ttl, string bor, string messageID, string flags[], string messagemeta)

This API is used to send a message to one or more Wickr clients. The "users" field may contain an array of 1 or more users to send the message to. The message will be sent to each user on a separate 1-to-1 conversation. So, if the API request "users" field contains 5 users then 5 messages will be sent, using the text from the "message" field.

The users array and message arguments are required, the remaining arguments are optional.

cmdSendRoomMessage(string vgroupid, string message, string ttl, string bor, string messageID, string flags[], string messagemeta)

This API is used to send a message to a secure room or group conversation. If you want to send a message to a secure room or a group conversation you will need to get the vGroupID associated with that conversation. To do that the vGroupID will be returned when you create the room/conversation using the appropriate API. Also, the get rooms API will return a list of known rooms that you can send to, the vGroupID is contained in the response.

The vgroupid and message arguments are required, the remaining arguments are optional.

cmdSend1to1Attachment(string users[], string filename, string displayname, string ttl, string bor, string messagemeta)

This API is used to send a file to one or more users. The file will be sent to each of the users in the input users list, via individual 1-to-1 conversations. The filename identifies a file that is located on the client system or a URL that identifies a remotely accessible file. If this is a local file, the file must be located in a location on the client that is accessible by the bot's client software running on the Wickr IO Docker image. The displayname field will be sent in the file transfer message.

The users array and filename arguments are required, the remaining arguments are optional.

cmdSendRoomAttachment(string vgroupid, string filename, string displayname, string ttl, string bor, string messagemeta)

This API is used to send a file to a secure room or group conversation. The file will be sent to the conversation associated with the input vgroupid. The filename identifies a file that is located on the client system or a URL that identifies a remotely accessible file. If this is a local file, the file must be located in a location on the client that is accessible by the bot's client software running on the Wickr IO Docker image. The displayname field will be sent in the file transfer message.

The vgroupid and filename arguments are required, the remaining arguments are optional.

cmdSendMessageUserNameFile(string fileName, string message, string ttl, string bor, string messageID, string flags[], string messagemeta)

This API is used to send a message to a list of Wickr clients contained in the input file. The "fileName" contains the full pathname of a file that is readable by the Wickr IO bot. The file contains a list of Wickr users, one per line in the file. The input message will be sent to each user on a separate 1-to-1 conversation. So, if the API request "fileName" file contains 5 users then 5 messages will be sent, using the text from the "message" field.

The fileName and message arguments are required, the remaining arguments are optional.

cmdSendAttachmentUserNameFile(string fileName, string attachment, string displayname, string ttl, string bor, string messageID, string messagemeta)

This API is used to send a file to a list of Wickr clients contained in the input file. The "fileName" contains the full pathname of a file that is readable by the Wickr IO bot. The file contains a list of Wickr users, one per line in the file. The input file identified by the "attachment" will be sent to each user on a separate 1-to-1 conversation. So, if the API request "fileName" file contains 5 users then 5 messages will be sent, using the file from the "attachment" field.

The fileName and attachment arguments are required, the remaining arguments are optional.

Network and Security Group Message APIs

These APIs are used to send messages and files to the users in a Wickr network or security group. Since Wickr bots can only transmit to clients in the same network, the Wickr network is the network that the bot is in. The Wickr bot can transmit to any of the security groups that are associated with the network it is associated with.

cmdSendNetworkMessage(string message, string ttl, string bor, string messageID, string flags[], string messagemeta)

This API is used to send a message to all of the Wickr clients in the bot client's Wickr network. The message will be sent to each user on a separate 1-to-1 conversation. So, if the associated network contains 100 users then 100 messages will be sent, using the text from the "message" field.

The message argument is required, the remaining arguments are optional.

cmdSendNetworkAttachment(string filename, string displayname, string ttl, string bor, string messageID, string message, string messagemeta)

This API is used to send a file to all of the Wickr clients in the bot client's Wickr network. The file will be sent to each of the users in the Wickr network via individual 1-to-1 conversations. The filename identifies a file that is located on the client system or a URL that identifies a remotely accessible file. If this is a local file, the file must be located in a location on the client that is accessible by the bot's client software running on the Wickr IO Docker image. The displayname field will be sent in the file transfer message.

The message field is used to also transmit a message, in addition to sending the attachment.

The filename argument is required, the remaining arguments are optional.

cmdSendNetworkVoiceMemo(string filename, string displayname, string ttl, string bor, string messageID, string message, string messagemeta)

This API is used to send a voice memo to all of the Wickr clients in the bot client's Wickr network. The voice memo will be sent to each of the users in the Wickr network via individual 1-to-1 conversations. The filename identifies a voice memo file that is located on the client system, this file must be located in a location on the client that is accessible by the bot's client software running on the Wickr IO Docker image. The displayname field will be sent in the file transfer message.

The message field is used to also transmit a message, in addition to sending the attachment.

The filename argument is required, the remaining arguments are optional.

cmdGetSecurityGroups(string page, string size)

This API will return a list of information for the Security Groups that are associated with the Wickr network the bot client is in. The page and size values are used to iterate through a large list of security groups. The page identifies which page of security groups to retrieve, where each page contains size number of entries. The page and size input values are optional but if specified they both have to be specified.

The value returned is a JSON array containing the following entries:

```
{
  "size" : <size of security group>,
  "name" : "<security group name>",
  "id" : "<security group ID>"
}
```

The "size" value is the number of users that are in the security group. The "name" is the actual name of the security group. The "id" is a unique identifier for the security group. The "id" value is used in the following APIs to send to security groups users.

cmdSendSecurityGroupMessage(string message, string groupids[], string ttl, string bor, string messageID, string flags[], string messagemeta)

This API is used to send a message to all of the Wickr clients in the security groups identified by the groupids value. The message will be sent to each user on a separate 1-to-1 conversation. So, if the associated security groups contain 100 users then 100 messages will be sent, using the text from the "message" field.

The message and groupids arguments are required, the remaining arguments are optional.

cmdSendSecurityGroupAttachment(string groupids[], string fileName, string displayname, string ttl, string bor, string messageID, string message, string messagemeta)

This API is used to send a file to all of the Wickr clients in the security groups identified by the groupids value. The file will be sent to each of the users via individual 1-to-1 conversations. The filename identifies a file that is located on the client system or a URL that identifies a remotely

accessible file. If this is a local file, the file must be located in a location on the client that is accessible by the bot's client software running on the Wickr IO Docker image. The displayname field will be sent in the file transfer message.

The message field is used to also transmit a message, in addition to sending the attachment.

The groupids and fileName arguments are required, the remaining arguments are optional.

cmdSendSecurityGroupVoiceMemo(string groupids[], string fileName, string displayname, string ttl, string bor, string messageID, string message, string messagemeta)

This API is used to send a voice memo to all of the Wickr clients in the security groups identified by the groupids value. The voice memo will be sent to each of the users via individual 1-to-1 conversations. The filename identifies a voice memo file that is located on the client system, this file must be located in a location on the client that is accessible by the bot's client software running on the Wickr IO Docker image. The displayname field will be sent in the file transfer message.

The message field is used to also transmit a message, in addition to sending the attachment.

The groupids and fileName arguments are required, the remaining arguments are optional.

Message Status APIs

These APIs are available to the bulk send APIs of the Wickr IO bot (i.e. network and security sending, file name list sending). The APIs will provide the ability to track the number of messages sent and remaining to be sent as well as if errors have occured during the sending of any of the messages. Errors will be identified on a per user basis as well, which can help determine if there was a problem sending to specific Wickr users.

cmdAddMessageID(string messageid, string sender, string target, string datesent, string message)

Before sending a message that you want to track you will need to call this API to add the message ID information to the Wickr IO client. This API is used to add a message ID entry to the Wickr IO clients database. The key part of this API is the message ID, it MUST be unique. The other values are determined by the integration using this API.

The messageid value should uniquely identify the message being sent.

The sender is a string that should be used to identify the sender of the message. The contents of this value is up to the integration using it. It can be used to restrict access to the message ID

information, so that users interacting with your integration cannot see message information for other users.

The target is a string that should be used to identify who the message(s) are being sent to. The contents of this value is up to the integration using it. The intent is to determine what type of message was being sent.

The datesent is used to identify the date and time when the message was sent. The contents of this value is up to the integration using it.

The message value is used to save the actual message or a string that identifies to the users the message that was sent.

cmdDeleteMessageID(string messageid)

This API will delete all entries in the Wickr bot's client database associated with the input message ID value.

cmdGetMessageIDEntry(string messageid)

This API will retrieve the information associated with the input message ID value. The value returned is a JSON object with the following format:

```
{
  "message_id" : "<message ID value>",
  "sender"     : "<sender value>",
  "target"     : "<target value>",
  "when_sent"  : "<when sent value>",
  "message"    : "<message value>"
}
```

cmdGetMessageIDTable(string page, string size)

This API is used to retrieve all of the message ID entries from the Wickr bot client's database. Since this table can get very large over time you will need to limit the number of entries retrieved by using the page and size values. The page is a 0 relative value used to identify which page to retrieve, where each page contains a number of entries equal to the size value. If the table is large enough you will need to iterate through each page until the number of entries is less than the size value. The return value from this API is a JSON array of the message ID objects, as shown in the cmdGetMessageIDEntry() API above.

cmdCancelMessageID(string messageID)

This API is used to cancel the transmit associated with the input messageID value. The bot client will attempt to stop the transmission. There is no guarantee that the transmit will be fully cancelled, but any subsequent transmits associated with the messageID should not occur.

cmdGetMessageStatus(string messageid, string type, string page, string size)

This API is used to retrieve the status of a specific message, identified by the input messageid value. The type value identifies what type of information should be returned. There are two types of message status values that can be returned:

- "full" for a full status for all users that the message is being sent to
- "summary" for a summary of the transmission for the associate message being sent.

The type value is optional, if not specified the default is to return a "summary" status of the associated message transmission. If the status to be returned is a "summary" then the following JSON object will be returned:

```
{
  "num2send" : <number of users to send to>,
  "pending"  : <number pending to send>,
  "sent"     : <number sent>,
  "failed"   : <number failed to send>,
  "acked"    : <number acked by receiver>
}
```

The values returned are all numbers. The "acked" value is the number of users that have responded to the bot. NOTE: a user sending a message to the bot will acknowledge all message ID entries associated with that user. This behavior will change in future versions.

If the type value is "full" then an array of values is returned, one for each user that the message is targeted to. The JSON array returned will contain objects with the following format:

```
{
  "user"           : "<user ID>",
  "status"         : <status of message to user>,
```

```
  "status_message" : "<error status if any>"
}
```

The "user" value is the user name that the message will be sent to. The "status" value is a number value that identifies the status of the message that is being sent to the "user". The "status" can have a value of one of the following:

- 0 means the message is pending to be sent to the associated user
- 1 means the message has been sent to the associated user
- 2 means the message failed to be sent to the associated user, see the "status_message" for details
- 3 means the message was sent and acknowledged by the associated user

If the "status" value returned for a user is a failed (3) then the "status_message" value will also be returned, otherwise the "status_message" value will not be returned.

The page and size input values are optional but should be used if the number of users the message is sent to is large. The page is a 0 relative value used to identify which page to retrieve, where each page contains a number of entries equal to the size value. If the table is large enough you will need to iterate through each page until the number of entries is less than the size value.

cmdSetMessageStatus(string messageid, string user, string status, string statusmessage)

This API can be used to modify the status and status message associated with a specific user's message status. The statusmessage value is optional. The messageid and user value will uniquely identify a message ID entry in the Wickr bot client's database. If there is no entry for the associated message ID and user then an entry will be created in the database.

The messageid value can have an empty string value (i.e. ""), which can be used to update ALL message ID entries for the specified user. This can be used for example to acknowledge all messages sent to that user. If the messageid value is empty and there are NO user entries for the user then nothing will be added to the database.

Key-Value APIs

These APIs provide the ability to save and retrieve values to/from a persistent encrypted storage location. The values will be stored in an encrypted database, that is associated with the Wickr IO client. Currently, only string values can be saved using these APIs.

These APIs do not have any relationship to the Wickr messaging, they are supplied for use by the Wickr IO integrations to have a way to save persistent data.

> **Warning**
>
> Since these values are stored in the Wickr IO client database, the values will be lost if the Wickr IO client database is reset. Also, the values are not accessible until the Wickr IO client is in the logged in state.

cmdAddKeyValue(string key, string value)

This API is used to save, or update the value associated with the input key.

cmdGetKeyValue(string key)

This API will return the string value associated with the input key.

cmdDeleteKeyValue(string key)

This API will delete the key-value information associated with the input key.

cmdClearAllKeyValues()

This API will clear (delete) all key-value pairs from the persistent storage.

Node.js Bot API (Development toolkit)

This section describes the Wickr IO Node.js Bot API framework and how to use it with several examples. This API provides tools for easier and more efficient development of Wickr IO integration bots. It utilizes the Wickr IO Node.js addon APIs to make it easier to develop integrations.

The Wickr IO Node.js Bot API is published to the default NPM registry. The name of the published module is `wickrio-bot-api`.

You will need to have a require() statement to include the Wickr Node.js Bot API in your integration. Also, since the Wickr IO addon is required you will need to have a require() statement for it as well. For example:

```
var addon = require('wickrio_addon');
```

```
var botapi = require('wickrio-bot-api');
```

start(client_username)

This API is used to start the connection with the Wickr IO client and get it ready for use. This API uses the Wickr IO addon `clientInit()`, `isConnected()` and `getClientState()` APIs to make sure the Wickr IO client connection is initialized, there is a valid connection and the client is in the `RUNNING` state. This is helpful since it may take the Wickr IO bot client longer to initialize than your integration.

If you use this API you will not need to call these addon APIs. This API will return true if successful, else returns false.

Parameters:

- client_username(REQUIRED) - The string user name of the Wickr IO client that is going to be used.

Example:

```
var result = botapi.start(process.argv[2]);
if (result === true) {
  console.log("Client started successfully");
} else {
  console.log("Client failed to start");
  process.exit();
}
```

startListening(callback)

This API initiates the asynchronous reception of received messages from the Wickr IO client. The passed callback function will be called whenever a message is received from the Wickr IO client.

Parameters:

- callback(REQUIRED) - The callback function that will be called when a message is received.

For more information, see the section called “Receive Asynchronous Messages”.

close()

This API will close the currently open connection to the Wickr IO client. This should be called when done interacting with the client set up by the start() API.

Example:

```
process.on('SIGINT', function() {
  console.log("Received SIGINT. Graceful shutdown ...");
  botapi.close();
  process.exit();
});
```

encryptEnv()

This API encrypts any environment variables that were input in the configuration part and were saved in the processes.json file, specifically any variables that contain sensitive information such as API Tokens (ex: Google App Client ID), or bot client specific variables such as a Bot Client Server.

loadData()

This API reads the encrypted user database from 'users.txt', which is an array of users personal information with their Wickr emails and any other information that was saved using the saveData() API. The user database is stored in the wickrUsers variable, which is an array of users personal information with their Wickr emails and any other information that was saved.

This API returns nothing.

saveData()

Encrypts the user database array contained in the wickrUsers variable, which is an array of users personal information with their Wickr emails and any other information that was saved, and saves it to the 'users.txt' file.

This API returns true if successful, else returns false.

addUser(wickrUser)

This API adds a user to the bot's user database array called wickrUsers, which is an array of users personal information with their Wickr emails and any other information that was saved.

Parameters:

- wickrUser(REQUIRED) - The WickrUser object to be added to the user database.

This API returns the user object after it is added to the user database.

parseMessage(message)

This API parses and breaks down an incoming received message from the client. If successful the parsed values will be returned in an object.

Parameters:

- message(REQUIRED) - The message object received from the Wickr IO client.

The returned object contains the following properties:

- command - The command portion of the message
- argument - The argument portion of the message
- message - The full message text
- sender - The sender of the message
- vgroupid - The group ID if sent to a group

For more information, see the section called “Receive Asynchronous Messages”.

getUser(userID)

This API searches the user database for the input user ID and returns the WickrUser object if found.

Parameters:

- userID(REQUIRED) - The string user ID of the user to be retrieved.

This API returns the user object if successful, else returns false.

getUsers()

This API is used to retrieve the entire user database.

Returns the bot's user database array.

deleteUser(userID)

Searches the user database for the passed user, deletes the user and returns the WickrUser object if successful, else returns false.

Parameters:

- userID(REQUIRED) - The string user ID of the user to be deleted.

getVersions(packageFile)

This API returns a string that contains the versions associated with all of the main components of the WickrIO environment. This includes the WickrIO client version, the WickrIO addon version, and the WickrIO bot API version.

Parameters:

- packageFile(REQUIRED) - The path to the package.json file for the integration.

This API returns a string containing version information for all WickrIO components.

Addon and Bot API Usage Examples

This section contains several examples of the use of the Wickr IO addon and the Wickr IO Bot API.

API Initialization

Before the Wickr IO Node.js addon API can be used you will need to initialize it in your JavaScript code. This initialization is done by calling the "start()" API (from the Wickr IO Bot APIs) and passing the client name associated with the Wickr IO client. For example:

```
const WickrIOAPI = require('wickrio_addon'); //WickrIO node.js addon which allows
 talking directly to our api
const WickrIOBotAPI = require('wickrio-bot-api'); //Development toolkit to help create
 bots/integrations
const WickrUser = WickrIOBotAPI.WickrUser;

var bot, tokens, bot_username, bot_client_port, bot_client_server;
var tokens = JSON.parse(process.env.tokens);

async function main() {
  try {
    bot_username = tokens.BOT_USERNAME.value;
    bot = new WickrIOBotAPI.WickrIOBot();
    var status = await bot.start(bot_username)
    if (!status)
      exitHandler(null, {
        exit: true,
        reason: 'Client not able to start'
```

```
      });
  } catch (err) {
    console.log(err);
  }
}
```

After the call to the "start()" API the client interface will be fully initialized. At this point you can start using the other APIs to communicate with the Wickr IO client.

Sending message to a room

The following code fragment shows the use of the `cmdSendRoomMessage()` API to send a message to a specific secure room. Before making the call you will need to get a valid vGroupID.

```
var msg = "Sorry, I'm not allowed to delete all the files in the directory.";
try {
    var sMessage = WickrIOAPI.cmdSendRoomMessage(vGroupID, msg);
    console.log(sMessage); //if successful should print "Sending message"
} catch(err){
    //Handle error here
    console.log(err);
}
```

Creating a room and sending an attachment

The following code shows the creation of a secure room and then sending a file to that room:

```
var members = [{ "name" : "username001" }, { "name" : "username002" }];
var moderators = [{ "name" : "username001" }, { "name" : "username002" }];
var bor = "600";  //OPTIONAL
var ttl = "1000"; //OPTIONAL
var title = "Example Room";
var description = "The Good Room";
var message = "Testing time!"
var attachmentURL = "https://www.alsop-louie.com/wp-content/uploads/2017/03/wickr-
logo-2-crop.png"
var displayname = "Logo.png";
try {
    var vGroupID = WickrIOAPI.cmdAddRoom(members, moderators, title, description, ttl,
 bor);
    //if successful should print a json with vgroupid of the newly created room
```

```
    console.log(vGroupID);
    //Notice: in this example the ttl and bor arguments are omitted and command will
 still work
    var cmd = WickrIOAPI.cmdSendRoomAttachment(vGroupID, attachmentURL, displayname);
    //if successful should print "Sending message"
    console.log(cmd);
} catch(err){
    //Handle errors here
    console.log(err);
}
```

Receive Asynchronous Messages

There are two types of messaging APIs supported by the Wickr IO Node.js addon:

- synchronous API calls: where a request is made to the Wickr IO client and a response is received
- asynchronous messaging: where you specify a callback function which the Wickr IO addon will call when a message is received. All synchronous APIs will wait for the response to return before proceeding.

The following code shows you how to initiate the asynchronous messaging and shows a callback function that will process the incoming messages.

```
await bot.startListening(listen); //Passes a callback function that will receive
 incoming messages into the bot client

async function listen(message) {
  try {
    var parsedMessage = bot.parseMessage(message); //Parses an incoming message and
 returns and object with command, argument, vGroupID and Sender fields
    if (!parsedMessage) {
      return;
    }
    console.log('parsedMessage:', parsedMessage);
    var wickrUser;
    var command = parsedMessage.command;
    var message = parsedMessage.message;
    var argument = parsedMessage.argument;
    var userEmail = parsedMessage.userEmail;
    var vGroupID = parsedMessage.vgroupid;
    var convoType = parsedMessage.convoType;
```

```
    var personal_vGroupID = "";
    if (convoType === 'personal')
      personal_vGroupID = vGroupID;
    var location = bot.findUser(userEmail); //Check if a user exists in the database
 and get his position in the database
    console.log('location:', location)
    if (location === -1) {
      wickrUser = new WickrUser(userEmail, {
        index: 0,
        personal_vGroupID: personal_vGroupID,
        command: "",
        argument: ""
      });
      var added = bot.addUser(wickrUser);
      console.log(added);
    }
    var user = bot.getUser(userEmail);
    user.token = "example_token_A1234";

    //how to determine the command a user sent and handling it
    if (command === '/help') {
      var reply = "What can I help you with?";
      var sMessage = WickrIOAPI.cmdSendRoomMessage(vGroupID, reply); //Respond back to
 the user or room with a message(using vGroupID)
      var users = [userEmail];
      var sMessage = WickrIOAPI.cmdSend1to1Message(users, reply); //Respond back to the
 user(using user wickrEmail)
      console.log(sMessage);
    }
  } catch (err) {
    console.log(err);
  }
}
```

The asynchronous messaging APIs will turn on or off the asynchronous reception of messages received by the Wickr IO client. After calling the "cmdStartAsyncRecvMessages(callback)" API, messages received will be sent to the callback function identified in that API. To turn off the asynchronous reception of messages use the "cmdStopAsyncRecvMessages()" API.

> **Note**
>
> Always add try/catch blocks for errors when calling addon APIs

Using the asynchronous messaging does require your program handles the background events associated with the reception of these messages. This can be tricky based on the single threaded nature of JavaScript.

API Shutdown

When you are done using the API you will need to shut it down. This is done by calling the "close()" Bot API. This will also stop the asynchronous message receiving for the bot.

```
process.stdin.resume(); //so the program will not close instantly

async function exitHandler(options, err) {
  var closed = await bot.close();
  console.log(closed);
  if (err) {
    console.log("Exit Error:", err);
    process.exit();
  }
  if (options.exit) {
    process.exit();
  } else if (options.pid) {
    process.kill(process.pid);
  }
}

//catches ctrl+c and stop.sh events
process.on('SIGINT', exitHandler.bind(null, {
  exit: true
}));

// catches "kill pid" (for example: nodemon restart)
process.on('SIGUSR1', exitHandler.bind(null, {
  pid: true
}));
process.on('SIGUSR2', exitHandler.bind(null, {
  pid: true
}));
```

Logging API

This section describes the logging module that can be imported from the WickrIOAPI. This logging module allows for different log levels and log file rotations in the bot integrations. Winston logger

is used as the logging library and the default log levels are those predefined by NPM and in order of importance are: error, warn, info, verbose, debug, and silly.

Getting Started with the Logger

To get started with the logger first import WickrLogger from the wickrio-bot-api library. Then you can access a new logger like so:

```
const logger = new WickrLogger().getLogger()
```

If you are currently using console.log as your default logger all instances of console.log can be piped through the logger with this block of code:

```
console.log = function () {
  logger.info(util.format.apply(null, arguments))
}
```

Similarly console.error can be piped through the logger like so:

```
console.error = function () {
  logger.error(util.format.apply(null, arguments))
}
```

Logger Configuration

The logger will look to the processes.json for the log level, max log file size, and max number of log files. These values can be changed by updating the values of LOG_LEVEL, LOG_FILE_SIZE, and LOG_MAX_FILES in the env section of processes.json.

Without modifying the log_tokens the log level will be set to info, the max file size will be set to 10MB and the max number of log files will be set to 5.

```
{
  "apps": [
    {
      "name": "WickrIO-Broadcast-Bot",
      "args": [],
      "script": "./build/index.js",
```

```
      "exec_interpreter": "node",
      "autorestart": false,
      "watch": ["package.json"],
      "ignore_watch": [".git"],
      "env": {
        "tokens": {},
        "log_tokens":
        {
          "LOG_LEVEL": "debug",
          "LOG_FILE_SIZE": "10m",
          "LOG_MAX_FILES": "5"
        }
      }
    }
  ]
}
```

Python Bot Development

To develop Wickr IO integrations in languages other than JavaScript such as Python, you will need to set up a Web REST API Interface integration on your machine and send HTTP/HTTPS requests to it. The following examples show how to do it in Python:

Set up your Python app

```
import requests
import json

URL = "http://localhost:4001/WickrIO/V1/Apps/CLIENT_API_KEY"
PARAMS = {'Accept': '*/*', 'Content-Type': 'application/json',
          'Authorization': 'Basic AUTH_KEY'}
```

Send 1-to-1 Message

```
data = {
    "message": "Welcome to AWS Wickr! This message will self-destruct eventually.",
    "users": [
        {"name": "exampleuser@wickr.com"}
```

```
    ]
}
sendMessage = requests.post(URL + "/Messages",
                            headers=PARAMS,
                            data=json.dumps(data))
print(sendMessage.content)
```

Add Room

```
data = {
    "room": {
        "title": "Security Group room for Sports in Bot Testing Network",
        "description": "Security Group room for Sports in Bot Testing Network",
        "ttl": "25536000",
        "bor": "0",
        "members": [
            {"name": "wickruser1@wickr.com"},
            {"name": "wickruser2@wickr.com"}
        ],
        "masters": [
            {"name": "wickruser1@wickr.com"}
        ]
    }
}
AddRoom = requests.post(URL + "/Rooms",
                        headers=PARAMS,
                        data=json.dumps(data))
json_data = json.loads(AddRoom.text)
room = json_data['vgroupid']
print(room)
```

Send Room message

```
data = {
    "message": "Welcome to AWS Wickr! This message will self-destruct eventually.",
    "vgroupid": "examplevgroupid"
}
sendMessage = requests.post(URL + "/Messages",
                            headers=PARAMS,
                            data=json.dumps(data))
```

```
print(sendMessage.content)
```

Get Statistics

```
getStatistics = requests.get(URL + "/Statistics",
                             headers=PARAMS)
print(getStatistics.json())
```

Delete Statistics

```
deleteStatistics = requests.delete(URL + "/Statistics",
                                   headers=PARAMS)
print(deleteStatistics.content)
```

Get Room

```
payload = {
    'vgroupid': room}
getRoom = requests.get(URL + "/Rooms/{0}".format(room),
                             headers=PARAMS)
print(getRoom.json())
```

Modify Room

```
data = {
        "title": "Modified room",
        "description": "Testing ModifyRoom command",
        "ttl": "25536000",
        "bor": "0"
}
modifyRoom = requests.post(URL + "/Rooms/{0}".format(room),
                           headers=PARAMS,
                           data=json.dumps(data))
print(modifyRoom.content)
```

Add Group Convo

```
data = {
    "groupconvo": {
        "members": [
            {"name": "exampleuser@wickr.com"},
            {"name": "exampleuser02@wickr.com"}
        ]
    }
}
AddGroupConvo = requests.post(URL + "/GroupConvo",
                              headers=PARAMS,
                              data=json.dumps(data))
print(AddGroupConvo.content)
response = json.loads(AddGroupConvo.text)
print(response['vgroupid'])
groupConvo = response['vgroupid']
print(groupConvo)
```

Get Group Convos (All)

```
getGroupConvos = requests.get(URL + "/GroupConvo",
                              headers=PARAMS)
print(getGroupConvos.json())
```

Get Group Convo (One)

```
getGroupConvo = requests.get(URL + "/GroupConvo/{0}".format(groupConvo),
                             headers=PARAMS)
print(getGroupConvo.json())
```

Delete Group Convo (One)

```
deleteGroupConvo = requests.delete(URL + "/GroupConvo/{0}".format(groupConvo),
                                   headers=PARAMS)
print(deleteGroupConvo.content)
```

Get Message

```
getMessage = requests.get(URL + "/Messages",
                          headers=PARAMS)
print(getMessage.json())
```

Set MsgRecvCallback

```
payload = {'callbackurl': 'http://localhost:8080/apps/callback'}
setMsgRecvCallback = requests.post(URL + "/MsgRecvCallback",
                                   headers=PARAMS,
                                   params=payload)
print(setMsgRecvCallback.content)
```

Get MsgRecvCallback

```
getMsgRecvCallback = requests.get(URL + "/MsgRecvCallback",
                                  headers=PARAMS)
print(getMsgRecvCallback.content)
```

Delete MsgRecvCallback

```
deleteMsgRecvCallback = requests.delete(URL + "/MsgRecvCallback",
                                        headers=PARAMS)
print(deleteMsgRecvCallback.content)
```

Complete Python Bot Example

```
import requests
import json
import time

URL = "http://localhost:4001/WickrIO/V1/Apps/CLIENT_API_KEY"
PARAMS = {'Accept': '*/*', 'Content-Type': 'application/json',
          'Authorization': 'Basic AUTH_KEY'}
```

```
def get_messages():
    """Get new messages from Wickr"""
    try:
        response = requests.get(URL + "/Messages", headers=PARAMS)
        return response.json()
    except Exception as e:
        print(f"Error getting messages: {e}")
        return []

def process_message(message):
    """Process a message from Wickr"""
    try:
        # Extract message content and sender
        content = message.get('message', '')
        sender = message.get('sender', '')

        print(f"Received message from {sender}: {content}")

        # Simple echo response
        if content:
            reply = {
                "message": f"You said: {content}",
                "users": [{"name": sender}]
            }

            response = requests.post(URL + "/Messages",
                                     headers=PARAMS,
                                     data=json.dumps(reply))
            print(f"Reply sent: {response.status_code}")
    except Exception as e:
        print(f"Error processing message: {e}")

def main():
    """Main bot loop"""
    print("Starting Python Wickr bot...")

    while True:
        # Get and process new messages
        messages = get_messages()
        for message in messages:
            process_message(message)

        # Wait before checking for new messages again
```

```
        time.sleep(5)

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

Automatic Configuration

As of the 5.116 release you can use AWS services to define the bot credentials, token values and other configuration information. You can use Wickr published docker images (i.e. bot-enterprise and bot-cloud) to start the bots. If you do use this method to automatically configure your bots you will not need to use the CLI to add the bots to your running docker image. All the credentials for the bots configured using this method will be secure in the AWS secrets manager service.

To use this method to configure your bots you will need to use the AWS_SECRET_NAME environment variable to identify the AWS secret that contains the configuration information.

Secrets Manager Value

The AWS_SECRET_NAME environment variable will identify an ARN that is used to access the specific secret which contains the configuration information needed to start the bots. The following is an example:

```
AWS_SECRET_NAME='arn:aws:secretsmanager:us-east-1:999999999999:secret:wickenterprise/
beta/my-test-bot-zZzZzz'
```

This secret contains the "wickr_config" key with the value being the configuration information needed to configure and start the bots on the docker image. The configuration information is stored in the secret as an escaped JSON string, for example the following is the plaintext secret value:

```
{"wickr_config":"{  \"clients\":[  {  \"integration\":\"wickrio-file-bot\",  \"name
\":\"user-file-bot\",  \"password\":\"password\",  \"configS3File\": {    \"key\" :
 \"configs_9-3-21/conf.wickr\",    \"bucket\" : \"bots-for-enterprise\",    \"region
\" : \"us-west-2\"  },  \"configPassword\":\"password\",  \"tokens\":[  {  \"name\":
\"CLIENT_NAME\",  \"value\":\"user-file-bot\"  },  {  \"name\":\"WICKRIO_BOT_NAME\",
 \"value\":\"user-file-bot\"  },  {  \"name\":\"DATABASE_ENCRYPTION_CHOICE\",  \"value
\":\"no\"  }  ]  }  ] }"}
```

The following is the un-escaped value for the "wickr_config" key, in the specified secret.

```
{
  "clients":[
    {
      "name":"user-file-bot",
      "password":"password",
      "integration":"wickrio-file-bot",
      "configS3File":{
        "key":"configs_9-3-21/conf.wickr",
        "bucket":"bots-for-enterprise",
        "region":"us-west-2"
      },
      "configPassword":"password",
      "tokens":[
        {
          "name":"CLIENT_NAME",
          "value":"user-file-bot"
        },
        {
          "name":"WICKRIO_BOT_NAME",
          "value":"user-file-bot"
        },
        {
          "name":"DATABASE_ENCRYPTION_CHOICE",
          "value":"no"
        }
      ]
    }
  ]
}
```

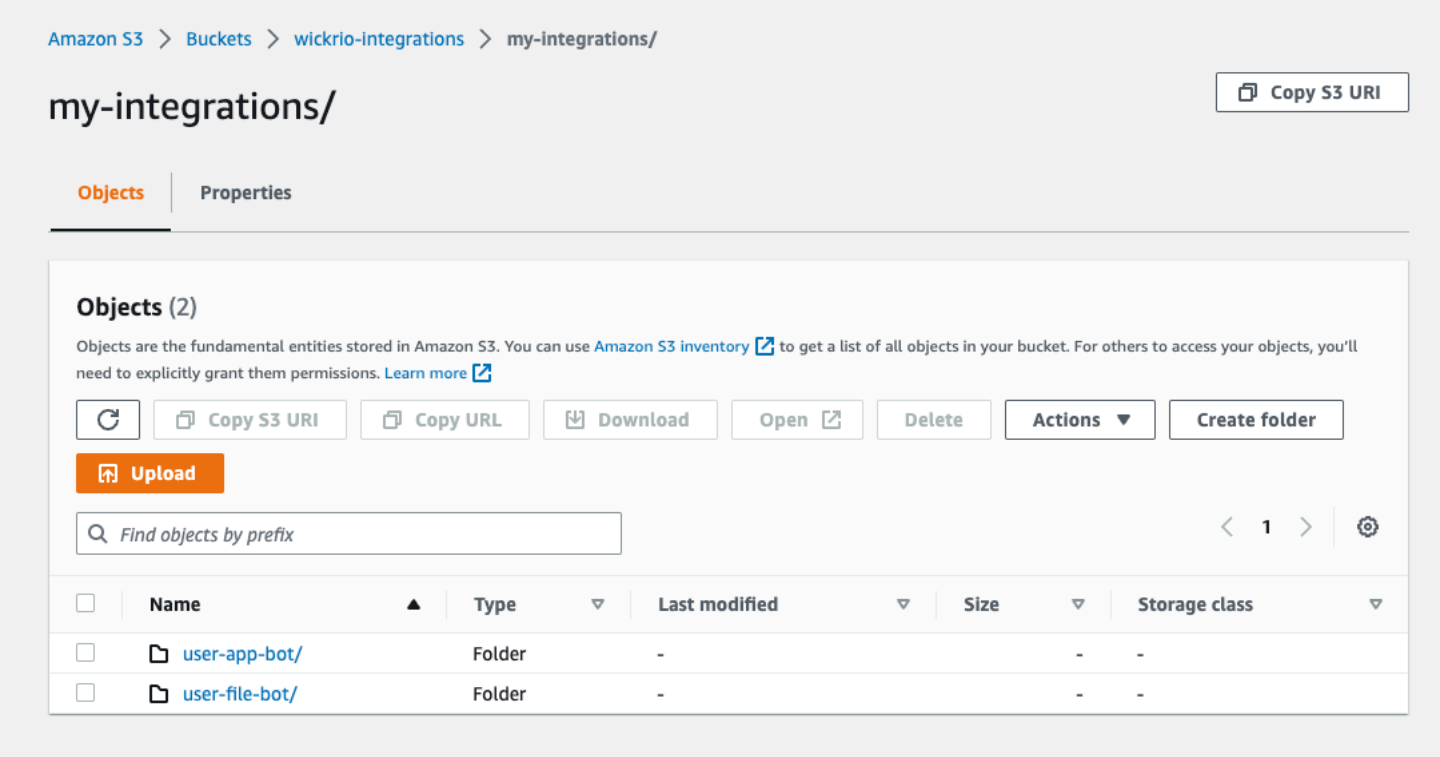
This is an example of an enterprise version, it contains the "configS3File and "configPassword" key/values which are needed to identify the conf.wickr file. The "key" value for the "configS3File" identifies the folder and filename for the config.wickr file. The "configPassword" identifies the password necessary to descrypt the config.wickr file. These values are not needed for bots running on the bot-cloud Docker images.

Using Custom Integrations

You can also use AWS S3 to load your own custom integrations. You will store them in an AWS S3 bucket, which can then be used by a Wickr IO docker image. The following environment variables will identify the S3 bucket and folder where these custom integrations will be located.

```
AWS_S3_INTEGRATIONS_REGION='us-east-1'
AWS_S3_INTEGRATIONS_FOLDER='test'
AWS_S3_INTEGRATIONS_BUCKET='wickrio-integrations'
```

The contents of the AWS S3 bucket/folder will contain one or more folders, one folder for each integration you want to be used by the Wickr IO bot. The name of the folder is used as the name of the integration that you will use to work with your bots. For example, see the image below, there are two folders in the bucket/folder. They are "user-app-bot" and "user-file-bot", which are the names of those two integrations. If the "integration" value in the "client" entry (see above) has the value "user-app-bot" or "user-file-bot" it will use the integration code from that folder.



The contents of each of the integration folders will be the software.tar.gz file that contains all of the integration files (see the section on developing your own custom bots).

Definitions

This section contains definitions of objects and message formats that are referenced by other parts of this site.

Topics

Wickr message formats

This section describes the format of the Wickr messages utilized by the Wickr IO addon APIs and the Wickr IO Web Interface REST messaging APIs. It explains the various types of Wickr messages you can encounter. Each message type includes a "msgtype" field and the following table lists the values associated with this field.

Message Type	msgtype
Text message	1000
Verification messages	3000
File transfer	6000
Calling messages	7000
Location message	8000
Edit message	9000
Edit reaction message	9100
Create room	4001
Modify room members	4002
Leave room	4003

Message Type	msgtype
Modify room parameters	4004
Delete room	4005
Delete message	4011
Message attributes message	4012
Message attributes sync request	4013
Modify private property	4014

All of the messages are represented using JSON. The following table describes the possible fields that are contained in the message JSON.

> **Note**
>
> The edit messages are only seen by the compliance bot installations (Wickr Enterprise) or by the data retention bot (AWS Wickr).

Field	Description
control	JSON object that defines the control message information. Contents described below.
file	JSON object that defines the details of a file transfer message. Contents described below.
id	Unique identifier for the message
message_id	The text associated with a text message
msgtype	Type of message, values defined in table above.

Field	Description
msg_ts	The time the message was sent, accurate to the microsecond.
receiver	Wickr ID of the receiver, for 1-to-1 messages.
sender	Wickr ID of the sending client.
sender_type	Indicates if this is a guest user or normal user.
time	Displayable time message was sent.
time_iso	The time in ISO format (YYYY-MM-DD hh:mm:ss.xxx)
vgroupid	The unique vGroupID of the conversation.

Text message

The msgtype for all text-based messages is 1000. Text-based messages can be sent either directly to the Wickr bot or in secure Room/Group conversations.

One-to-one messages

The following shows a one-to-one text message format:

```
{
"message_id":"3960e020ca4211e799802f2894564caa",
"message":"This is a typical 1:1 message",
"msg_ts":"1510777143.738976",
"msgtype":1000,
"receiver":"user001",
"sender_type": "normal",
"sender":"user003",
"time": "7/11/23 5:19 PM",
"time_iso": "2023-07-11 17:19:58.781",
"ttl": "7/10/24 5:19 PM",
"vgroupid":"fb6e21630c05fde50ae39113c3626018712cf2c374b4a80eba4d28ced9419c07"
}
```

Text messages with links

If you send a text message that contains links, and the security group settings have the "Send Link Preview" option enabled, the text message will contain a list of the URLs for those links:


```
{
"links":[
{
"url":"https://testdaily.com/image/test-laughing/"
}
],
"message":"Check out this link https://testdaily.com/image/test-laughing/",
"message_id":"fb7d7d20b25d11eb9a2d77f565346d8b",
"msg_ts":"1620740228.594362",
"msgtype":1000,
"receiver":"bnuser02@userworld.com",
"sender":"bnuser01@userworld.com",
"time":"5/11/21 1:37 PM",
"ttl":"6/10/21 1:37 PM",
"vgroupid":"2c0ae523d2b1af3e43af80b5fafec05548fd2e33fee4c021c66033c6416bb6bb"
}
```


Group and Room conversation messages

The following shows a normal group or secure room conversation text message format:


```
{
"message_id":"3960e020ca4211e799802f2894564caa",
"message":"This is a typical 1:1 message",
"msg_ts":"1510777143.738976",
"msgtype":1000,
"sender_type": "normal",
"sender":"user003",
"time": "7/11/23 5:19 PM",
"time_iso": "2023-07-11 17:19:58.781",
"ttl": "7/10/24 5:19 PM",
"vgroupid":"Sa6783e427e164d37f2e8177874ee192523e6cc9520416bf96850ca01730bf07"
}
```

Note

In some cases, the Wickr IO client does not track the list of clients associated with group conversations, so the list of destination clients will not be included. You can use the supplied Wickr IO APIs to retrieve the members associated with a secure room or group conversation vGroupID.

File transfer messages

The msgtype for file transfer messages is 6000. This message type contains information about a file transfer message. The "file" JSON object contains the details of the file being transferred, described in this table:

Field	Description
filename	The display name of the file being transferred.
guid	A unique identifier for the transferred file.
uploadedbyuser	The user who uploaded the file
uploadedtimestamp	Upload time
localfilename	The full path name of the file on the Wickr IO Gateway system.

Files received by the Wickr IO client will be decrypted and remain on the Wickr IO client until removed by your software.

The files sent for screen shots will be identified by a **isscreenshot** key value pair, in the "file" object. This is a Boolean value, where true identifies the file as a screenshot. If the **isscreenshot** key is not found then the file is not a screen shot.

One-to-one messages

The following shows the format of a file transfer message in 1:1 conversations:

```
 {
    "file": {
        "filename": "picture.jpeg",
        "guid": "AD20D048-9B60-4F32-A691-2D4BE4152E58",
        "localfilename": "/opt/WickrIO/clients/compliancebot01/attachments/
attachment_20171116111610865_picture.jpeg",
        "uploadedbyuser": "cn0623_01@amazon.com",
        "uploadedtimestamp": "7/11/23 5:22 PM"
    },
    "message_id": "91a189c0cae911e79ec4eb19a763225b",
    "msg_ts": "1510849017.756174",
    "msgtype": 6000,
    "receiver":"user001",
    "sender": "user003",
    "sender_type": "normal",
    "time": "7/11/23 5:22 PM",
    "time_iso": "2023-07-11 17:22:02.348",
    "vgroupid": "53042f1bd04491c6f3732a871e27ab516a8d1534cc1e2d25c4e4869ce72e8541"
}
```

Group and Room conversation messages

The following shows the format of a file transfer message in group or room conversation conversations:

```
 {
    "file": {
        "filename": "picture.jpeg",
        "guid": "AD20D048-9B60-4F32-A691-2D4BE4152E58",
        "localfilename": "/opt/WickrIO/clients/compliancebot01/attachments/
attachment_20171116111610865_picture.jpeg",
        "uploadedbyuser": "cn0623_01@amazon.com",
        "uploadedtimestamp": "7/11/23 5:22 PM"
    },
    "message_id": "91a189c0cae911e79ec4eb19a763225b",
    "msg_ts": "1510849017.756174",
    "msgtype": 6000,
    "sender": "user003",
    "sender_type": "normal",
    "time": "7/11/23 5:22 PM",
    "time_iso": "2023-07-11 17:22:02.348",
    "vgroupid": "S3042f1bd04491c6f3732a871e27ab516a8d1534cc1e2d25c4e4869ce72e8541"
```

```
}
```

Calling messages

The `msgtype` for all location type messages is 7000 Calling messages will have a **call** object with a subset of the following values:

Name	Description
calluri	The URI associated with the call.
calluriipv6	The IPv6 URI associated with the call
duration	The call duration in seconds. Sent in the end of call message. The duration of a call in a room starts immediately regardless if any users answer the call. The duration of a call in a one-on-one conversation will begin when the called user answers the call.
invitemsgid	The message ID associated with the original call start message. This is sent when another user is added to a call.
meetingid	The unique meeting ID associated with the call.
messagetype	The type of call message this is, see table below.
participants	List of username hash values for all of the potential participants of the call.
startmsgid	The message ID associated with the call start message.
status	The current state of the call (i.e. starting, completed)

Name	Description
version	The call protocol version
versioncheck	Boolean value to check call protool version

Call in a room

The following is an example of a message format when a call is started in a room:

```
{
call: {
calluri: '44.211.195.26:16504',
calluriipv6: '[2610:1f18:68b5:a01:c6e2:93fa:b7ae:b934]:16504',
meetingid: 'ba20e10f-9476-40b9-9c6d-46c03ed54a45',
messagetype: 0,
participants: [
'8bf491c9a3b14117f0553a3b48b325b7abat5438757e96e539c77810c59d1c33',
'09fc89173d0538487f0ac2ac593a6421b6cfdec7443cedc1b115a63d3ed2ebb1',
'122ac8391c2009305ce4369e4ad8ad4d5ed258b9870345571c114672235b482f'
],
status: 0,
version: 2,
versioncheck: true,
vgroupid: 'Sa6783e427e164d37f2e8177874ee192523e6cc9520416bf96850ca01730bf00'
},
message_id: '5aa8a300353711f0b014adea6236a0b1',
msg_ts: '1747717230.896467',
msgtype: 7000,
receiver: 'yaybot',
respond_api: 'http:///0/Apps//Messages',
sender: 'guptabde@amazon.com',
sender_type: 'normal',
time: '5/20/25 5:00 AM',
time_iso: '2025-05-20 05:00:30.896',
ttl: '5/20/26 5:00 AM',
vgroupid: 'Sa6783e427e164d37f2e8177874ee192523e6cc9520416bf96850ca01730bf00'
}
```

Note

The following different type of states of the call messages are only seen by the compliance bot installations (Wickr Enterprise) or by Data Retention bot (AWS Wickr).

The **status** fields identifies the current state of the call, the following table identifies what the **status** values are:

Call Status	Status Value
Call starting	0
Call completed	1
Call missed	2
Call cancelled	3

The **messagetype** identifies the message type of the call message, the following table identifies the **messagetype** values:

Call Message Type	Value	Description
Start call	0	Starting a call
End call	1	Ending a call
Missed call	2	Missed a call request
Declined call	3	Declined a call request
Silent ring	4	Sent from a device when it answers a call. The other devices for that client will stop ringing.

When a user answers a call you will see the following messages:

```
{
"call":{
"status":0
},
"message_id":"23a4b710ed2d11eab7697d766fcb32a2",
"msg_ts":"1599058871.990588",
"msgtype":7000,
"sender":"cbtestuser@wickr.com",
"time":"9/2/20 3:01 PM",
"vgroupid":"S49bf359b1229270fdbbc9fbca6289ce1f2171bf9f278c7b37cd3a76ab12e2e1"
}
```


When the call is done you will see the following message to end the call:


```
{
"call":{
"status":1
},
"message_id":"52ec4ab0ed2d11eabecd817847e86976",
"msg_ts":"1599058950.990589",
"msgtype":7000,
"sender":"cbtestuserthree@wickr.com",
"time":"9/2/20 3:02 PM",
"vgroupid":"S49bf359b1229270fdbbc9fbca6289ce1f2171bf9f278c7b37cd3a76ab12e2e1"
}
```


Adding user to a call

During a call it may be necessary to add a user to a call. The following sequence will show a normal call started with two users, and then a third user is added to the call. Please note, the call is started on a specific conversation identified by the "vgroupid" value (in this case the vgroup ID is "S4666b353873113884feb66d1409875a81b40aa5c0ddbab040ec11f1b38e752c"). When the additional user is added you will see the "vgroupid" associated with the one-on-one conversation to that new user (in this case the vgroup ID is "4c84cfa0a7b780f76fdb8d86cb5569f61e66a7a43ef8ab1cc5b537b427e1989b").

The following is the start call message, it only includes the original two participants:

```
{
"call":{
"meetingid":"177539f4-7d60-4c52-8f1c-d98f421e847f",
"status":0
},
"message_id":"3af57350ed3711eab7697d766fcb32a2",
"msg_ts":"1599063205.990632",
"msgtype":7000,
"sender":"cbtestuserthree@wickr.com",
"time":"9/2/20 4:13 PM",
"vgroupid":"S4666b353873113884feb66d1409875a81b40aa5c0ddbab040ec11f1b38e752c"
}
```

The following message is the called user accepting the call:

```
{
"call":{
"status":0
},
"message_id":"4068dde0ed3711eabecd817847e86976",
"msg_ts":"1599063214.990632",
"msgtype":7000,
"sender":"cbtestusertwo@wickr.com",
"time":"9/2/20 4:13 PM",
"vgroupid":"S4666b353873113884feb66d1409875a81b40aa5c0ddbab040ec11f1b38e752c"
}
```

This message is sent when a new participant is added to the call. Notice the different "vgroupid" value in the message. The "meetingid" is the same as the running call's "meetingid":

```
{
"call":{
"meetingid":"177539f4-7d60-4c52-8f1c-d98f421e847f",
"status":0
},
"message_id":"526beb90ed3711eab7697d766fcb32a2",
"msg_ts":"1599063245.990632",
```

```
"msgtype":7000,
"receiver":"cbtestuser@wickr.com",
"sender":"cbtestuserthree@wickr.com",
"time":"9/2/20 4:14 PM",
"vgroupid":"4c84cfa0a7b780f76fdb8d86cb5569f61e66a7a43ef8ab1cc5b537b427e1989b"
}
```

The following message is sent from the new participant when accepting the call:

```
{
"call":{
"status":0
},
"message_id":"54ebdfb0ed3711eab7697d766fcb32a2",
"msg_ts":"1599063249.990632",
"msgtype":7000,
"receiver":"cbtestuserthree@wickr.com",
"sender":"cbtestuser@wickr.com",
"time":"9/2/20 4:14 PM",
"vgroupid":"4c84cfa0a7b780f76fdb8d86cb5569f61e66a7a43ef8ab1cc5b537b427e1989b"
}
```

This message is sent to the original "vgroupid" when the call is ended:

```
{
"call":{
"status":1
},
"message_id":"5bbf0150ed3711eab7697d766fcb32a2",
"msg_ts":"1599063260.990632",
"msgtype":7000,
"sender":"cbtestuserthree@wickr.com",
"time":"9/2/20 4:14 PM",
"vgroupid":"S4666b353873113884feb66d1409875a81b40aa5c0ddbab040ec11f1b38e752c"
}
```

This message is also sent to the one-on-one **vgroupid** for the invited user when the call is ended:

```
{
"call":{
"status":1
},
"message_id":"5bc59100ed3711eab7697d766fcb32a2",
"msg_ts":"1599063260.990632",
"msgtype":7000,
"receiver":"cbtestuser@wickr.com",
"sender":"cbtestuserthree@wickr.com",
"time":"9/2/20 4:14 PM",
"vgroupid":"4c84cfa0a7b780f76fdb8d86cb5569f61e66a7a43ef8ab1cc5b537b427e1989b"
}
```


Missed call

If a call is attempted on a one-to-one conversation and the target participant does not answer the call you will see a message with a missed call **messagetype** value of 2. This message is sent from the user originating the call to the called user. There is no duration associated with this call and should not be considered a completed call.


```
{
"call":{
"status":2
},
"message_id":"fa0a5d80ed3e11eab7697d766fcb32a2",
"msg_ts":"1599066532.990665",
"msgtype":7000,
"receiver":"cbtestusertwo@wickr.com",
"sender":"cbtestuserthree@wickr.com",
"time":"9/2/20 5:08 PM",
"vgroupid":"ec6cfd71ccb4034c9d77263da8c28d01d19a6f91746d3b9b61b868d0663008a4"
}
```


Declined calls

If a user on a one-on-one call selects to ignore the call, a declined call message will be seen. You should not see this type of message on room conversations.

```
{
"call":{
"status":3
},
"message_id":"1739df60ed4011eabecd817847e86976",
"msg_ts":"1599067011.990670",
"msgtype":7000,
"receiver":"cbtestuserthree@wickr.com",
"sender":"cbtestusertwo@wickr.com",
"time":"9/2/20 5:16 PM",
"vgroupid":"4c84cfa0a7b780f76fdb8d86cb5569f61e66a7a43ef8ab1cc5b537b427e1989b"
}
```


Location messages

The **msgtype** for all location type messages is 8000. This message is sent when a user sends their location in a conversation. The location will contain the user's latitude and longitude.

One-to-one messages

The following shows the format of a location type message in 1:1 conversations:


```
{
"location":{
"latitude":45.75017899435506,
"longitude":-74.99449803034105
},
"message_id":"1f88fdc08bec11ea81b689d23fa72c7b",
"msg_ts":"1588365684.583407",
"msgtype":8000,
"receiver":"user003",
"sender":"user100",
"sender_type": "normal",
"time": "7/11/23 5:33 PM",
"time_iso": "2023-07-11 17:33:24.394",
"ttl": "7/10/24 5:33 PM",
"vgroupid":"4ebf561eb2214c4e6f924d09e37bf80b6f9b85cb96b72badb03753d9ed26f7f4"
}
```

Group and Room conversation messages

The following shows the format of location type message in group or room conversation conversations:

```
{
"location":{
"latitude":45.75017899435506,
"longitude":-74.99449803034105
},
"message_id":"1f88fdc08bec11ea81b689d23fa72c7b",
"msg_ts":"1588365684.583407",
"msgtype":8000,
"sender":"user100",
"sender_type": "normal",
"time": "7/11/23 5:33 PM",
"time_iso": "2023-07-11 17:33:24.394",
"ttl": "7/10/24 5:33 PM",
"vgroupid":"Sebf561eb2214c4e6f924d09e37bf80b6f9b85cb96b72badb03753d9ed26f7f4"
}
```

Edit messages

> **Note**
>
> The edit messages are only seen by the compliance bot installations (Wickr Enterprise) or by data retention bot (AWS Wickr).

There are currently two types of edit messages supported, the location and the text types. The location type of edit message is sent when a user is sharing their location with someone else.

```
{
"edit":{
"type":"location",
"shareexpiriation":"";
"latitude":45.75017899435506,
"longitude":-78.99449803034105
```

```
},
"message_id":"1f88fdc08bec11ea81b689d23fa72c7b",
"msg_ts":"1588365684.583407",
"msgtype":9000,
"receiver":"user003",
"sender":"user100",
"sender_type": "normal",
"time": "7/11/23 5:30 PM",
"time_iso": "2023-07-11 17:30:15.103",
"ttl": "7/10/24 5:30 PM",
"vgroupid":"4ebf561eb2214c4e6f924d09e37bf80b6f9b85cb96b72badb03753d9ed26f7f4"
}
```

The text type of edit message is sent when the user sends a message that includes links in it. For example the user sends a message with the link https://howdoyoudo.com in it, the following is what the edit message would look like:

```
{
"edit":{
"originalmessageid":"11457fa08da211ea881baffab0b42745",
"text":"https://howdoyoudo.com",
"type":"text"
},
"message_id":"1163e5b08da211eab775a5032a0322ca",
"msg_ts":"1588553780.419871",
"msgtype":9000,
"sender":"user001@amazon.com",
"sender_type": "normal",
"time": "7/11/23 5:30 PM",
"time_iso": "2023-07-11 17:30:15.103",
"ttl": "7/10/24 5:30 PM",
"vgroupid":"S243f2ec645d3961bdd531f51f3244205d292b8d0fbd41802827746271d31d41"
}
```

If you send a text message with a link and the security group has the "Send Link Preview" option enabled, the edit message may contain an array of links information and link image meta information. The following shows these additional fields:

```
    {
    "edit":{
    "linkimagemeta":{
    "domain":"userworld.com",
    "guid":"c43c71ef-4373-444e-a22d-dfce34d38a7a",

 "hash":"32bc71721bea8a456a06f364995012d3ebfc41aaaad0c2dd632b0f0bae4690f4bffcd5a4c23b4af16086fc
    "key":"00a85d12214f596d4eac929d82287cccdead2a00c542f850322c1655494be2a40d"
    },
    "links":[
    {
    "faviconurl":"https://testdaily.com/favicon.ico",
    "imageurl":"https://testdaily.com/uploads/gallery/test-laughing.gif",
    "pagetitle":"Test laughing",
    "sitename":"Test Daily",
    "url":"https://testdaily.com/gallery/image/Test-laughing/"
    }
    ],
    "originalmessageid":"fb7d7d20b25d11eb9a2d77f565346d8b",
    "text":"https://twinsdaily.com/gallery/image/2234-burns-laughing/",
    "type":"text"
    },
    "message_id":"fe9c4950b25d11eb9a2d77f565346d8b",
    "msg_ts":"1620740233.829096",
    "msgtype":9000,
    "receiver":"bnuser02@userworld.com",
    "sender":"bnuser01@userworld.com",
    "sender_type": "normal",
    "time": "7/11/23 5:35 PM",
    "time_iso": "2023-07-11 17:35:41.411",
    "ttl": "7/10/24 5:35 PM",
    "vgroupid":"2c0ae523d2b1af3e43af80b5fafec05548fd2e33fee4c021c66033c6416bb6bb"
    }
```


Edit content messages

Edit content messages are sent when a user edits the contents of a previously sent message. The Edit Content message will contain the message ID associated with the original message and the text of the updated message. The original message text will not be included. This message type was introduced in the 5.92 version of the WickrIO software. The Edit Content messages are used to

identify when the text of a message is edited as well as when the links contained in a message are edited.

The following is a basic example of an Edit Content message where the text is edited:

```
{
"content_edited": true,
"edit":{
"type":"edit_content",
"originalmessageid":"36028e2025dd11ec9cdafd3f2bfa110f",
"text":"This is the edited message"
},
"message_id":"3fcef29025dd11ec9cdafd3f2bfa110f",
"msg_ts":"1633439273.17562",
"msgtype":9000,
"receiver":"user001+comp9321_01@wickr.com",
"sender":"user001+comp9321_02@wickr.com",
"sender_type": "normal",
"time": "7/11/23 5:35 PM",
"time_iso": "2023-07-11 17:35:41.411",
"ttl": "7/10/24 5:35 PM",
"vgroupid":"56e3b0570daad62b2e2d14db8d33632f6175514022183a042660c7b8901dec79"
}
```

The **type** value, within the **edit** group, identifies this as an Edit Content message. The **originalmessageid** identifies the message ID of the original message. The **text** field is the new value of the message.

If the original message contains links, and the "Send Link Previews" option is set for the security group, there will be two Edit Content messages sent. One of these messages is associated with the text message changes and another that will contain the link image meta information. The Edit Content message that contains the "content_edited" with a true value is associated with the message text, as seen below:

```
{
"content_edited":true,
"edit":{
"type":"edit_content",
"originalmessageid":"f4bddd80255e11ec8c960b356f9f1aad",
```

```
      "text":"This is a test with https://wickr.com",
      "links":[
      { "url":"https://wickr.com" }
      ],
      },
      "message_id":"b17cffb0264911ec9cdafd3f2bfa110f",
      "msg_ts":"1633485849.387294",
      "msgtype":9000,
      "receiver":"user001+comp9321_01@wickr.com",
      "sender":"user001+comp9321_02@wickr.com",
      "sender_type": "normal",
      "time": "7/11/23 5:35 PM",
      "time_iso": "2023-07-11 17:35:41.411",
      "ttl": "7/10/24 5:35 PM",
      "vgroupid":"56e3b0570daad62b2e2d14db8d33632f6175514022183a042660c7b8901dec79"
      }
```

The following edit content message is associated with the links that are in the message:

```
      {
      "edit":{
      "type":"edit_content",
      "originalmessageid":"f4bddd80255e11ec8c960b356f9f1aad",
      "text":"This is a test with https://wickr.com",
      "linkimagemeta":{
      "domain":"wickr.com",
      "guid":"4747a757-32db-4748-8a59-f5fc02cf811b",

 "hash":"4848c43ba0ac0cca685cd3053076198f6d710ed02fa7adf4822ba752e48c5328b7bc947d6e0499bfca6d83
      "key":"0016c4902a79160966335053c07d96accd048ca5ce858f9def2364d11c36b7f345"
      },
      "links":[
      {
      "description":"Wickr provides end-to-end encrypted messaging, audio calling,
 video conferencing, file and location sharing, and more.",
      "faviconurl":"https://wickr.com/favicon.ico",
      "imageurl":"https://wickr.com/wp-content/uploads/2020/12/wickr-pro-
screens-4-1.png",
      "pagetitle":"Home",
      "sitename":"Wickr",
      "url":"https://wickr.com"
```

```
    }
    ]
    },
    "message_id":"b270fca0264911ec9cdafd3f2bfa110f",
    "msg_ts":"1633485850.986206",
    "msgtype":9000,
    "receiver":"user001+comp9321_01@wickr.com",
    "sender":"user001+comp9321_02@wickr.com",
    "sender_type": "normal",
    "time": "7/11/23 5:35 PM",
    "time_iso": "2023-07-11 17:35:41.411",
    "ttl": "7/10/24 5:35 PM",
    "vgroupid":"56e3b0570daad62b2e2d14db8d33632f6175514022183a042660c7b8901dec79"
    }
```


Edit reaction messages

> **Note**
>
> The edit messages are only seen by the compliance bot installations (Wickr Enterprise) or by Data Retention bot (AWS Wickr).

The edit reaction messages are used to modify the reactions associated with a message. The message layout is similar to the Edit message layout, but the **msgtype** value is 9100. The **edit** object contains the following fields:

Field Name	Description
originalmessageid	The message ID of the message that the reaction is associated
reactAdded	This is a boolean that is true if the reaction is added and false if removed
reaction	This is the reaction to enable or disable
type	This is always the value **reaction**

The "sender" field will identify who is adding or removing a reaction. The client should display the reaction and which users were associated with that reaction.

The following is a sample of adding a reaction to a message in 1:1 conversation:

```
{
    "edit":{
      "originalmessageid":"a1c00830ad3911ebaa04213b1ad0a9b2",
      "reactAdded":true,
      "reaction":"#",
      "type":"reaction"
    },
    "message_id":"b6d3d1b0add011ebb00f1b2dbb810704",
    "msg_ts":"1620239749.707944",
    "msgtype":9100,
    "receiver":"bn0523_bcast_bot",
    "sender":"bnuser01@userworld.com",
    "sender_type": "normal",
    "time": "7/11/23 5:35 PM",
    "time_iso": "2023-07-11 17:35:41.411",
    "ttl": "7/10/24 5:35 PM",
    "vgroupid":"4b32d7c8c6c37cc9e9506e9ed98ce37f0a96e1e45fcd4ca6f6d00c9d435d82e3"
}
```

The following is a sample of removing a reaction to a message:

```
{
    "edit":{
      "originalmessageid":"a1c00830ad3911ebaa04213b1ad0a9b2",
      "reactAdded":false,
      "reaction":"#",
      "type":"reaction"
    },
    "message_id":"68596ac0add211ebb00f1b2dbb810704",
    "msg_ts":"1620240477.36364",
    "msgtype":9100,
    "receiver":"bn0523_bcast_bot",
    "sender":"bnuser01@userworld.com",
    "sender_type": "normal",
    "time": "7/11/23 5:35 PM",
```

```
    "time_iso": "2023-07-11 17:35:41.411",
    "ttl": "7/10/24 5:35 PM",
    "vgroupid":"4b32d7c8c6c37cc9e9506e9ed98ce37f0a96e1e45fcd4ca6f6d00c9d435d82e3"
}
```

Wickr control messages

Wickr control messages are used to setup and configure the Wickr group and secure room conversations. These control messages will also be passed on to the integration software. You can use these control messages to construct and maintain the list of users that are part of group and secure conversations. The **control** JSON object contains the specific control message fields, as described in this table:

Field	Description
bor	The burn on read time in seconds.
changemask	A number that is the sum of values associated with which fields are being changed by the control message. Table below defines these values.
description	The description of the secure room conversation.
masters	Array of Wickr IDs that are moderators for the secure room conversation. In group conversations all members should be moderators.
members	Array of Wickr IDs that are members of the group or secure room conversation.
title	The title of the secure room conversation
ttl	The title of the secure room conversation

The `changemask` value is a number value that is created by adding the following flag values, based on what fields are contained in the **control** object:

Field	Value
Masters field	1
Time to live	2
Time filed	4
Description	8
Meeting ID key	16
Burn on read	32
File vault info	64

The following sections contain examples of control messages, note the values of the `changemask` fields.

Create room control message

The create room control message is used to create a group or secure room conversation. The following depicts a typical create room control message:

```
{
    "control":{
        "bor":0,
        "changemask":47,
        "description":"",
        "masters":["user001", "user002"],
        "members":["user001", "user002", "user003"],
        "msgtype":4001,
        "title":"Creating a room",
        "ttl":2592000
    },
    "message_id":"be452b00f89711e883588d1e7a946847",
    "msg_ts":"1544019125.75323",
    "msgtype":4001,
    "sender":"user002",
    "sender_type": "normal",
```

```
    "time": "7/11/23 5:12 PM",
    "time_iso": "2023-07-11 17:12:45.168",
    "ttl": "7/10/24 5:12 PM",
    "vgroupid":"S58a15186365d2125a9b417e71b99bcb29e3770078e157e953cfbe28443eb750"
}
```

In the example above you will see the `changemask` value 47 which is equal to the sum of the "burn on read" (32), "masters field" (1), "time to live" (2), "title field" (4) and the "description" (8). The changemask values are used in other control messages as well.

Modify room members control message

The modify room member control message is used to modify the list of users associated with a group or secure room conversation. Members can be added or removed from the conversation using this control message.

```
{
    "control":{
        "addedusers":[],
        "deletedusers":["testbot"],
        "msgtype":4002
    },
    "message_id":"d34058a0f89711e88760d7c8037ea946",
    "msg_ts":"1544019160.275884",
    "msgtype":4002,
    "sender":"user002",
    "sender_type": "normal",
    "time": "7/11/23 5:15 PM",
    "time_iso": "2023-07-11 17:15:49.079",
    "ttl": "7/10/24 5:15 PM",
    "vgroupid":"S58a15186365d2125a9b417e71b99bcb29e3770078e157e953cfbe28443eb750"
}
```

The `addedusers` field identifies the list of users that were added to the conversation. The `deletedusers` field identifies the list of users that were removed from the conversation.

Note

This control message has a fixed set of fields and does not require the `changemask` field.

Leave room control message

The leave room control message is sent when a user is leaving a secure room or group conversation. The sender is the user leaving the conversation:

```
{
    "message_id":"f660fe80f89711e887d86d198d2ef374",
    "msg_ts":"1544019219.210107",
    "msgtype":4003,
    "sender":"user002",
    "sender_type": "normal",
    "time": "7/11/23 5:54 PM",
    "time_iso": "2023-07-11 17:54:31.688",
    "ttl": "7/10/24 5:54 PM",
    "vgroupid":"S58a15186365d2125a9b417e71b99bcb29e3770078e157e953cfbe28443eb750"
}
```

This control message is a simple control message and does not contain the "control" JSON object. The `msgtype` fields identifies the leave room action.

Modify room parameters control message

The modify room parameters control message is used to modify one or more fields associated with a group or secure room conversation.

```
{
    "control":{
        "bor":0,
        "changemask":47,
        "description":"change description",
        "masters":["user001", "user002"],
        "members":["user002","user002"],
        "msgtype":4004,
        "title":"Creating a room",
```

```
        "ttl":2592000
    },
    "message_id":"db805750f89711e8a01ab328ac0b2f04",
    "msg_ts":"1544019174.117057",
    "msgtype":4004,
    "sender":"user002",
    "sender_type": "normal",
    "time": "7/11/23 5:59 PM",
    "time_iso": "2023-07-11 17:59:57.473",
    "ttl": "7/10/24 5:59 PM",
    "vgroupid":"S58a15186365d2125a9b417e71b99bcb29e3770078e157e953cfbe28443eb750"
}
```

This message is similar to the create room control message. The `changemask` field has the same definition as well.

Delete room control message

The delete room control message is used to delete a specific group or secure room conversation.

```
{
    "message_id":"06364710f89811e899418b6723464a0c",
    "msg_ts":"1544019245.773676",
    "msgtype":4005,
    "sender":"user002",
    "sender_type": "normal",
    "time": "7/11/23 6:03 PM",
    "time_iso": "2023-07-11 18:03:01.100",
    "ttl": "7/12/23 6:03 PM",
    "vgroupid":"S7879eb406958d83b991a5f2acb29e5ad8565a4faa41e1c5cbd7004c5586ddd5"
}
```

This control message does not have a "control" JSON object. The `msgtype` identifies that action to be performed.

Delete message

Here is the delete control message that will delete a message from all of the specified sender's devices. The sender is the sender of the control message, not the original sender of the message

being deleted. The message will not be deleted from other users devices: The message to delete is identified by the id value in the control group.

```
{
    "control":
    {
        "message_id":"a59a3520ca2f11e7a14cd1c8bf2a1be8",
        "isrecall":false,
        "msgtype":4011
    },
    "message_id":"aa4a3580ca2f11e7a156a34c720bab3d",
    "msg_ts":"1510769172.735225",
    "msgtype":4011,
    "sender":"user003",
    "time":"11/15/17 1:06 PM",
    "vgroupid":"Sb0e9297f2208dc86b63b288df8c226882e1052b65022edb9edb9ecf6e77db08"
}
```

Here is the delete control message that will recall the message from all users the message was sent to:

```
{
    "control":
    {
        "isrecall":true,
        "message_id":"c2855a10ca2f11e7afa15b0943d8c736",
        "msgtype":4011
    },
    "message_id":"c5bce5e0ca2f11e78946112c51861afa",
    "msg_ts":"1510769218.785332",
    "msgtype":4011,
    "sender":"user003",
    "time": "7/11/23 6:04 PM",
    "time_iso": "2023-07-11 18:04:03.772",
    "ttl": "7/10/24 6:04 PM",
    "vgroupid":"Sb0e9297f2208dc86b63b288df8c226882e1052b65022edb9edb9ecf6e77db08"
}
```

Modify private property

This message identifies when a conversation is pinned or un-pinned. The "pinned" value will be true when the conversation is pinned, and "false" when it is being un-pinned.

```
{
    "control":
    {
        "pinned":true,
        "msgtype":4014
    },
    "message_id":"c5bce5e0ca2f11e78946112c51861afa",
    "msg_ts":"1510769218.785332",
    "msgtype":4014,
    "sender":"user003",
    "sender_type": "normal",
    "time": "7/11/23 6:07 PM",
    "time_iso": "2023-07-11 18:07:21.981",
    "ttl": "7/10/24 6:07 PM",
    "vgroupid":"Sb0e9297f2208dc86b63b288df8c226882e1052b65022edb9edb9ecf6e77db08"
}
```

This message identifies when a message is starred:

```
{
    "control": {
      "attributes": [
        {
          "isstarred": true,
          "msgid": "74f16f3011d511ee9456414dec7866b3"
        }
      ],
      "msgtype": 4012
    },
    "message_id": "7a38aa00201511eeb723c79384d92c63",
    "msg_ts": "1689098711.200122",
    "msgtype": 4012,
    "sender": "cn0623_01@amazon.com",
    "sender_type": "normal",
    "time": "7/11/23 6:05 PM",
```

```
    "time_iso": "2023-07-11 18:05:11.200",
    "ttl": "7/10/24 6:05 PM",
    "vgroupid": "S5d76ccc43c2bb3bc93e7273798ce8f4b89bfa8429b33888fbcd454e5d233a19"
}
```

Text message meta data

As of the 5.81 release of WickrIO, support was added to record the text message meta data fields. These fields are used to define GUI widgets specific to the text message being transmitted. Specific GUI widgets include buttons and list items. If you are interested in creating bots with buttons or lists, refer to the transmit message arguments section of the Node.js Addon APIs.

Text message table meta data

The following text message is a broadcast bots response to a /report command. This sample message contains two meta objects, the "tablemeta" object and the "textcutmeta" object. The "tablemeta" object contains information used to display the list on the Wickr client. The column name objects define the headers for the table. The "items" array contains the entries for the table. The "textcutmeta" array contains information used to remove parts of the "message" value when displaying the message to the user. This is done because the text of the "message" may be redundant to the information contained in the "tablemeta" object. This will allow Wickr clients that do not yet support the "tablemeta" protocol yet.

```
{
   "message":"Here are the past 5 broadcast message(s):\n(1) this is a broadcast\n(2)
 Send to one user\n(3) This is the 3rd broadcast\n(4) Hello World!\n(5) Test one\nTo
 get started, select the broadcast for which you would like to generate a report",
   "message_id":"80cff660a75a11eb986e31ce44694ab3",
   "msg_ts":"1619529271.494966",
   "msgtype":1000,
   "receiver":"bnuser01@userworld.com",
   "sender":"bn0523_bcast_bot",
   "tablemeta":{
      "actioncolumnname":"Select",
      "firstcolumnname":"Message",
      "items":[
         {
            "firstcolumnvalue":"this is a broadcast",
```

```
            "response":"1"
         },
         {
            "firstcolumnvalue":"Send to one user",
            "response":"2"
         },
         {
            "firstcolumnvalue":"This is the 3rd broadcast",
            "response":"3"
         },
         {
            "firstcolumnvalue":"Hello World!",
            "response":"4"
         },
         {
            "firstcolumnvalue":"Test one",
            "response":"5"
         }
      ],
      "name":"List of Sent Broadcasts"
   },
   "textcutmeta":[
      {
         "end":146,
         "start":0
      }
   ],
   "sender_type": "normal",
   "time": "7/11/23 6:07 PM",
   "time_iso": "2023-07-11 18:07:21.981",
   "ttl": "7/10/24 6:07 PM",
   "vgroupid":"4b32d7c8c6c37cc9e9506e9ed98ce37f0a96e1e45fcd4ca6f6d00c9d435d82e3"
}
```


The text message above is a sample, and does not include all of the optional fields for the meta data you may see in a message. The "tablemeta" object can contain the following objects:

Name	Description
name	This object is a string that is the name of the table.

Name	Description
firstcolumnname	This object is a string that is the name of the first column of the table.
secondcolumnname	This object is a string that is the name of the second column of the table. This is optional.
actioncolumnname	This object is a string that is the name of the action column of the table. This is optional.
items	This is an array of entries for the table.

The items is an array and can contain 0 or more entries for the table. Each entry in this table will contain the following objects:

Name	Description
firstcolumnvalue	This object is a string that is the value for the first column.
secondcolumnvalue	This object is a string that is the value for the second column. This is optional.
response	This object is a string that is the value to be returned to the bot when the list is selected. This is optional.

The "textcutmeta" is used to maintain some backwards compatability with Wickr clients that do not support lists. The "textcutmeta" is a list that identifies text characters that will be removed from the message text when the client supports table meta data. Typically the text that is removed is when the list information is in the message text, see the example above. The fields of the each of the "textcutmeta" entries identifies the starting index and ending index of what characters to cut from the message text. These values are 0 relative for the first character in the message text.

Text message button meta data

As of the 5.81 release of WickrIO, support was added for buttons. Wickr clients that support these buttons will display the buttons when the message is received from the bot. There are several types of buttons, but selecting a button will perform a specific operation, including sending a message to the bot or sending your location to the bot. The following is an example of a text message with button meta data:

```
{
   "buttonmeta":[
      {
         "msgbutton":{
            "message":"/ack",
            "text":"/Ack"
         }
      },
      {
         "locationbutton":{
            "locationtype":0,
            "text":"/Ack with Location"
         }
      }
   ],
   "message":"this is a broadcast\n\nBroadcast message sent by: bnuser01@userworld.com
\n\nPlease acknowledge message by replying with /ack",
   "message_id":"744dc0c0a75a11eb986e31ce44694ab3",
   "msg_ts":"1619529250.508240",
   "msgtype":1000,
   "receiver":"bnuser01@userworld.com",
   "sender":"bn0523_bcast_bot",
   "sender_type": "normal",
   "time": "7/11/23 6:07 PM",
   "time_iso": "2023-07-11 18:07:21.981",
   "ttl": "7/10/24 6:07 PM",
   "vgroupid":"4b32d7c8c6c37cc9e9506e9ed98ce37f0a96e1e45fcd4ca6f6d00c9d435d82e3"
}
```

The "buttonmeta" array contains all of the buttons associated with this text message. There are three types of buttons supported:

Name	Description
msgbutton	This is a normal button type that contains the text of the button and the message to be sent to the bot when the button is selected.
locationbutton	This is a location button that will perform a location based operation. Currently, the "locationtype" value of 0 means the client will send the location of the device to the bot when the button is selected. There will be other location type buttons supported in the future.

The "msgbutton" object will have the following objects:

Name	Description
text	This is the text of the button to be displayed
message	This is the message that is sent to the bot when the button is selected.

The "locationbutton" type performs a location type of operation. The "locationbutton" object has the following objects:

Name	Description
text	This is the text of the button to be displayed
type	The type of location operation to perform. Value of 0 is to have the client send the location to the bot when the button is selected. Other location button types will have the option to start and stop location sharing, not implemented yet.

Wickr IO Command Line Interface (CLI)

The Wickr IO command line interface commands are used to perform operations on the Wickr IO clients and the integrations you associate with them. The commands are described as follows:

Many commands require a client number parameter shown as `[<#>]`. To obtain the client number for a specific Wickr IO client, use the `list` command, which displays all configured clients with their corresponding numbers in the first column (e.g., 0, 1, 2). Use this number when executing commands that require the `[<#>]` parameter.

Topics

General Commands

The following are the general Wickr IO command line interface commands:

Command	Description
`help [command]`	Display the list of commands or help for the specified command.
`version`	Displays the current version number of the running Wickr IO Gateway.
`welcome [on|off]`	Displays the welcome message or changes whether the welcome message is displayed when you enter the command line interface.

Client Management Commands

The following are the Wickr IO command line interface client commands:

Command	Description
`add`	Adds a Wickr bot client to the Wickr IO Gateway. The Wickr bot client will be associated with a Wickr user account and any associated integration.
`avatar [<#>]`	This command will change the avatar associated with a Wickr IO client. The avatar file must be accessible to the software running on the Wickr IO docker image.
`config [<#>]`	Re-runs the configuration of the specified Wickr IO Client. Can only be done on paused clients.
`delete [<#>]`	Deletes an existing client from the Wickr IO Gateway. The Wickr bot client will no longer be associated with the Wickr IO Gateway system after the delete is completed.
`events [<#>]`	Displays the list of events associated with the specified client. These events are indications of when the client was started or stopped.
`list [integration]`	Display a list of Wickr clients that have been added to the Wickr IO Gateway. If the integration option is used, then a list of available integrations will be displayed. The integration commands are described in the integration's commands below.
`modify [<#>]`	Modify the settings of a client. You will be prompted for the settings that can be changed. You can change the password, the integration associated with the client, and/or change the settings.
`pause [<#>]`	Pause a running Wickr IO client.
`restart [<#>]`	Performs a stop and start of the Wickr IO client.
`start [<#>]`	Starts a Wickr IO client. If this is the first time the client has been started, you will be prompted for the password associate d with the Wickr user account.

Integration Management Commands

The following are the integration commands:

Command	Description
`export [<#>]`	Export the integration software associated with the specified Wickr IO client. This will create a tar file that can be used with other Wickr IO clients.
`import`	Import custom integration software. The integration software has to adhere to specific requirements, which are described later.
`list [integration]`	The list command with the 'integration' option will display a list of integrations that are available for use with Wickr IO clients. This command may take up to a minute to complete.
`rename [<#>]`	Rename the integration of the specified Wickr IO client. This is useful when you want to create a new integration using an existing integration as the base for the new integration.
`upgrade [<#>]`	Update integration software for a Wickr IO client. This command will check to see if there is a newer version of the integration software available and upgrade to that version.

Wickr IO clients logging

The Wickr IO bots generate log and output files that can be used to determine possible issues. These files can be accessed on your host where the Wickr IO docker container is running.

In case of any issues, these files should be sent to Wickr Support to allow them to diagnose any issues that cannot be easily fixed.

> **Note**
>
> Output and log files have a maximum file size. When this size is exceeded, a new file is created. The system keeps up to 5 output and log files on disk.

Logs are generated from three primary components: client provisioning logs, client logs, and integration logs.

Topics

Wickr IO client provisioning logs

Provisioning logs are generated when a new or existing bot client is added to the running Docker container using the docker CLI. To set up a bot client, see[Quick Start section].

Depending on how you setup the shared files, the provisioning log and output files are located in the following location:

`/opt/WickrIO/logs`

Below is an example of a different shared host volume while starting the docker container.

```
docker run -v /opt/WickrIOOnHost:/opt/WickrIO -ti public.ecr.aws/x3s2s6k3/wickrio/bot-cloud:latest
```

In the example above, the host directory shared with the Docker container is `/opt/WickrIOOnHost`. As a result,the logs will be located at: `/opt/WickrIOOnHost/logs`.

This directory contains several files:

- `WickrIOProvisionLog.output` - Contains detailed information useful for diagnosing issues during Wickr IO client provisioning.
- `WickrIOSvr.output` - Helps identify problems that may occur when starting a Wickr IO client.

Wickr IO client logs

Once the Wickr IO Client bot is successfully provisioned, the bot client and the background services initiate logging for all interactions with Wickr backend, along with internal system-level processes.

Depending on how you setup the shared files, the provisioning log and output files are located in the following location:

`/opt/WickrIO/clients/(client name)/logs`

Below is an example of a different shared host volume while starting the docker container.

```
docker run -v /opt/WickrIOOnHost:/opt/WickrIO -ti public.ecr.aws/x3s2s6k3/wickrio/bot-cloud:latest
```

In the above example, the host directory path shared with the docker container is `/opt/WickrIOOnHost`, then the logs path will be `/opt/WickrIOOnHost/clients/(client name)/logs`.

There are several files found in that directory. The file with the ".output" extension contains the most information and is useful in diagnosing issues with the Wickr IO client.

To view logs in real time, run the following command from the `/opt/WickrIO/clients/(client name)/logs` directory:

`tail -f (OutputFileName)`.

Wickr IO integration logs

Wickr IO integrations also generate log data, which can be helpful for diagnosing issues specific to the integration type used by the Wickr IO client.

Integration logs are located in the integration-specific directory associated with the Wickr IO client. For example, a broadcast bot named "test_bot" is running, its integration logs can be found at:

```
/opt/WickrIO/clients/test_bot/integration/wickrio-broadcast-bot/logs
```

Typical Wickr IO integrations will write output to a file named "log.output" and error output will be written to a file named "err.output". Output in the "err.output" is an indication the integration has crashed.

Wickr IO clients troubleshooting

Topics

These are troubleshooting suggestions associated with running the Wickr IO gateway, clients, and integrations. You may also run into some issues when developing your own custom Wickr IO integrations.

This section will describe some possible issues you may run into while using the Wickr IO client and the associated services

> **Note**
>
> AWS Support can help with AWS Wickr service configuration, network settings, and the Wickr bot SDK APIs. For issues with custom bot logic, container infrastructure, or host environment, contact your internal development or operations team.

Bot runtime architecture

Each Wickr IO bot has three layers, which AWS recommends independent monitoring for:

1. **Host** – The Amazon EC2 instance or on-premises server running Docker.
2. **Container** – The Wickr IO Docker container (`wickrio/bot-cloud` or `wickrio/bot-cloud-govcloud`).
3. **Bot process** – Your Node.js integration running inside the container.

The container can report as running while the bot process inside it has crashed or lost its connection to the Wickr network. Effective monitoring covers all three layers.

Setting up Wickr IO Docker container

Cannot load Docker image

If you receive the following error then you may have to make sure you have permissions to access the unix socket to communicate with the Docker engine:

`docker: Got permission denied while trying to connect to the Docker daemon socket at unix:...`

Solution:

For help setting the appropriate permissions for you Docker account, see Linux post-installation steps for Docker Engine

Wickr IO Fails startup due to directory permissions

When running the Docker container on a Mac system, you may encounter a failure when starting the Wickr IO Docker container if you have not changed the write permissions for the `/opt` directory. This issue typically arises after the `./start.sh` command. You will see the following errors:

```
  "QIODevice::write (QFile, "/opt/WickrIO/logs/WickrIOSvr.output"): device not open
terminate called after throwing an instance of 'zmq::error_t'
what(): No such file or directory
./start.sh: line 3: 7 Aborted WickrIOSvr"
```

This issue has only been seen on Mac systems. It occurs when a user creates the `/opt` directory as the root, and then runs the container as a user.

Solution:

This is caused by a permission issue in the `/opt` directory. To resolve the issue, change the permissions of the `/opt/WickrIO` directory (or the name of the directory you created for your installation) to make it writeable. The following command will fix this problem:

```
chmod 777 /opt/WickrIO
```

Host system running out of disk space

Your system is low on storage, preventing you from downloading and running a new Docker container. This issue may arise if you have not removed old Docker containers from your system.

Solution:

- Run `docker ps` to see all running containers.
- Run `docker kill container_id` on a container you would like to stop using.
- Run `docker rm container_id` to remove it from your system completely.

> **Note**
>
> Your integrations will not be affected and will stay on your system in your `/opt/WickrIO` directory for future use.

Docker Mounts denied Error

You're getting the following error: `docker: Error response from daemon: Mounts denied: The path /opt/WickrIO is not shared from OS X and is not known to Docker.`

Solution:

Restart Docker.

Wickr IO Not working on Mac M1 hosts

The Wickr IO Docker images are not built to run on the Mac M1 hosts. During our testing of the Mac M1 hardware, we discovered an issue related to the interaction exec() commands, which led to failures when trying to start a bot from the command line interface (CLI).

Solution:

If you want to run the Wickr IO Docker images on Mac M1 hardware, follow these steps:

- Make sure you have a version of Docker that supports the Linux/amd64 platform. The version we recommend for Mac is 20.10.14.
- When using the docker run command, make sure to include the —platform=linux/amd64 option

Provisioning Wickr IO client

Bot client provisioning failures

When adding a bot client to a Docker image, you may encounter issues during the provisioning step. This can occur for both new and existing clients. Below are some common problems you might face:

Username does not exist:

```
Enter command:add
Enter the user name:demobot
Enter the password:***********
Creating user:  "demobot"

Begin registration with password.

Begin register new user context.

Validation Error

Failed to create or login new user!

 demobot does not seem to exist. Please verify in the Admin Console.!
Enter command:
```

Solution

Make sure the bot username is correct and exists in the Wickr Admin Console.

Incorrect password:

```
Enter command:add
Enter the user name:deefrabot
Enter the password:******************
Creating user:  "deefrabot"

Begin registration with password.

Begin register new user context.

Begin register existing user context.

Either the username or password you entered was invalid.

Failed to create or login new user!

 User deefrabot exists already, password entered seems to be invalid!
Do you want to try a new password? (default: yes):
```

Solution

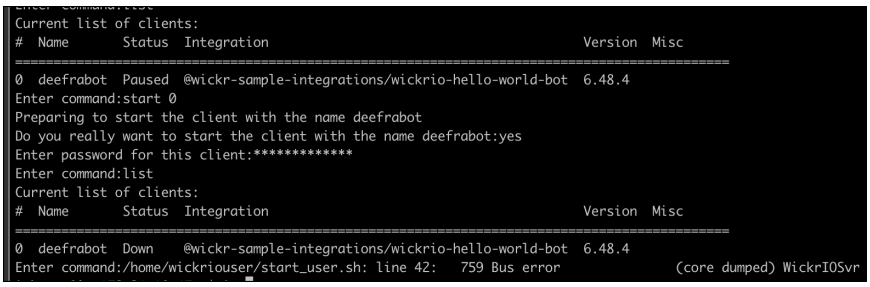
Make sure the password entered is correct. It should match the one used when creating the bot user in the Wickr Admin Console.

> **Note**
>
> If your issue is different from the scenarios defined above, you can also identify the issue by looking at the provisioning logs found in the following file: `/opt/WickrIO/WickrIOProvisionLog.output`
> For further assistance, Wickr support is always available to assist you with any issues that are not addressed here.

Start bot client failures

Incorrect password

When trying to start a bot client, if you enter an incorrect password, you will get a “Bus Error” and see the Docker container has exited.



Solution

- Remove the stopped container using `docker rm -f (container-name).`
- Set up the docker container again by sharing a different host volume.

Wickr IO client does not start

If your Wickr IO client isn't starting and the previous solutions haven't resolved the issue, try the following steps. First, check if the client is running by using the `ps` command. Enter the following command in the terminal:

```
ps -aef | grep wickrio_bot
```

This command should return an entry for each Wickr IO client that is currently running. Additionally, the command line interface of the Wickr IO Docker container includes various commands that can help you diagnose any client issues.

- Enter the `list` command to see the list of Wickr IO clients.
- If the client state is `Paused` then use the `start` command to start the client.
- If the client state is `Running` and there are still no associated process running, then check the `WickrIOSvr.output` file for the background services to see if there is any issue starting the client. For more information on logging details, see Wickr IO clients logging.

Wickr IO command line interface

Wickr IO client is not running.

Once the client has started, it should remain in the `Running` state unless it is manually paused or deleted. If, at any point, your client enters a `Down` or `Not Running` state unexpectedly, follow the solution below.

Solution:

Try restarting the Wickr IO client using the `restart` command. If the issue continues, check the provisioning and client logs for more information, and then contact Wickr Support.

Turn on debugging in Wickr IO CLI

If you are facing issues with performing any operation using the Wickr IO command line interface like adding or deploying a custom bot , this procedure will help you view more detailed debugging information on the command line interface.

Solution

To enable debugging, enter debug in the CLI.

> **Note**
>
> This action will output a lot of debug information with this option enabled.

```
Enter command:debug on
Debug output is now ON, it will remain ON after restart as well!
Enter command:
```

Client and Integration compatibility issues

With the release of new Wickr IO Docker images containing changes to the Wickr IO Add-on APIs, when you upgrade your container to use the new version, it is possible there will be compatibility issues that will cause Integrations to not work.

Solution

If you are running an officially Wickr supported integration, you will see the **Needs Upgrade** text on your Wickr IO client.

To upgrade your integration to avoid any compatibility issues, complete the following steps.

1. Pause the running client using `pause (index)` command.
2. Upgrade the client using `upgrade (index)` command.
3. Start the client again.

If you are using a custom integration, make sure to update your integration to ensure compatibility with the updated Wickr IO container.

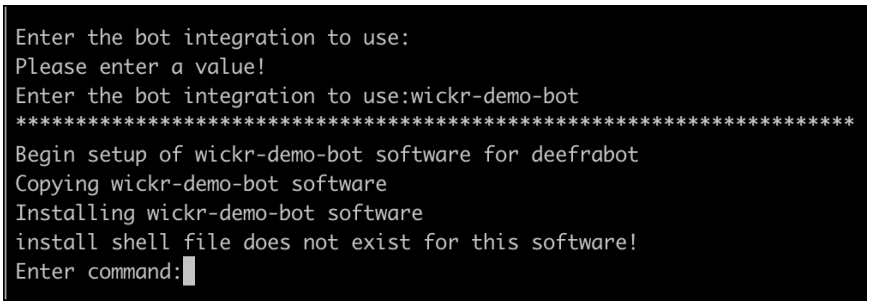
If you need to contact Wickr support it's helpful to have the version number of the Docker container ready. You can find the version number by using the `version` command in the Wickr

IO command line interface. Additionally, new versions of the Wickr IO Docker image will display the version numbers of the integrations when you run the `list` command. When addressing integration issues, having the version numbers for the integrations will also be beneficial.

Deploying custom Integrations

Cannot import a custom integration

If the necessary files are missing from the software.tar.gz imported into the Docker container, you will receive an "install shell file does not exist" error when starting a Wickr IO client using your custom integration.



Solution

Make sure all the required files are present in the zipped integration software imported in the docker container. You can use any sample integrations as reference.

Monitor bot health with Amazon CloudWatch

The following table summarizes the monitoring approaches for each layer, from simplest to most comprehensive.

Layer	What it catches	Effort	Reliability
Amazon EC2 status check	Host failure, hardware issues	None (built-in)	High
Container metric	Docker crash, OOM kill, restart loop	Low	Medium
Log-based metric filter	Container crash, restart loop (container output only)	Low	Low

Layer	What it catches	Effort	Reliability
Heartbeat metric	All of the above, plus silent disconnects and event loop blocks	Medium	High

For production bots, we recommend implementing all four approaches. They are not redundant – each catches different failure modes.

Monitor the host instance

Use the built-in Amazon EC2 `StatusCheckFailed` metric to detect host-level failures. On supported instance types, Amazon EC2 simplified automatic recovery is enabled by default and automatically migrates the instance to healthy hardware when a status check fails. The alarm below is informational – it notifies you that a recovery occurred, but no manual action is required. For more information, see Recover your instance in the *Amazon EC2 User Guide*.

```
HostDownAlarm:
  Type: AWS::CloudWatch::Alarm
  Properties:
    AlarmName: !Sub "${BotName}-HostDown"
    Namespace: AWS/EC2
    MetricName: StatusCheckFailed
    Dimensions:
      - Name: InstanceId
        Value: !Ref BotInstance
    Statistic: Maximum
    Period: 60
    EvaluationPeriods: 2
    Threshold: 1
    ComparisonOperator: GreaterThanOrEqualToThreshold
    AlarmActions:
      - !Ref AlertTopic
```

Monitor the Docker container

Create a script on the host that publishes a custom CloudWatch metric based on container status. Run it every minute using `cron`. For example:

```
#!/bin/bash
# /opt/wickr-bot/monitor.sh
CONTAINER_NAME="${1:-wickr-bot}"
NAMESPACE="WickrIO/Bots"

STATUS=$(docker inspect --format='{{.State.Running}}' "$CONTAINER_NAME" 2>/dev/null)

aws cloudwatch put-metric-data \
  --namespace "$NAMESPACE" \
  --metric-name ContainerRunning \
  --dimensions BotName="$CONTAINER_NAME" \
  --value "$([ "$STATUS" = "true" ] && echo 1 || echo 0)" \
  --unit Count
```

Add a cron entry to run the script:

```
* * * * * /opt/wickr-bot/monitor.sh container-name
```

Create an alarm that fires when the value drops to 0.

Monitor with log-based metric filters

If you forward container logs to Amazon CloudWatch Logs using the `awslogs` driver, you can create a metric filter that counts log events. When the container stops producing output, the alarm fires.

1. Open the Amazon CloudWatch console and navigate to **Log groups**.
2. Select your bot's log group (`/wickr/bots/bot-name`).
3. Choose **Metric filters**, then **Create metric filter**.
4. For **Filter pattern**, enter a single space (`" "`) to match any log line.
5. Set the metric namespace to `WickrIO/Bots` and the metric name to `BotLogEvents`.
6. Create an alarm on this metric: `Sum < 1` over a 5-minute period.

Add a heartbeat metric (recommended)

A heartbeat metric is the most reliable approach because it proves the Node.js process is alive and the event loop is not blocked. Add the following to your bot's entry point, after the bot starts successfully. For example:

```
const { CloudWatchClient, PutMetricDataCommand } = require('@aws-sdk/client-
cloudwatch')
const cw = new CloudWatchClient()
const BOT_NAME = process.env.BOT_NAME || 'wickr-bot'

setInterval(async () => {
  try {
    await cw.send(new PutMetricDataCommand({
      Namespace: 'WickrIO/Bots',
      MetricData: [{
        MetricName: 'Heartbeat',
        Dimensions: [{ Name: 'BotName', Value: BOT_NAME }],
        Value: 1,
        Unit: 'Count'
      }]
    }))
  } catch (e) { console.error('Heartbeat publish failed:', e.message) }
}, 60000)
```

Add the `@aws-sdk/client-cloudwatch` dependency to your bot's `package.json`.

Create an alarm on the heartbeat metric. For example:

```
BotProcessAlarm:
  Type: AWS::CloudWatch::Alarm
  Properties:
    AlarmName: !Sub "${BotName}-ProcessDown"
    Namespace: WickrIO/Bots
    MetricName: Heartbeat
    Dimensions:
      - Name: BotName
        Value: !Ref BotName
    Statistic: Sum
    Period: 300
    EvaluationPeriods: 2
    Threshold: 1
    ComparisonOperator: LessThanThreshold
    TreatMissingData: breaching
    AlarmActions:
      - !Ref AlertTopic
```

Set `TreatMissingData` to `breaching` so the alarm fires when the metric stops arriving entirely, rather than entering `INSUFFICIENT_DATA`.

Monitor memory usage

Node.js bots that maintain session state, cache data, or process file attachments can develop memory leaks over time. If the bot's memory usage exceeds the Docker container's memory limit, Docker terminates the container with an out-of-memory (OOM) kill. The container restarts automatically if you configured the `--restart` policy, but active user sessions and in-flight messages are lost.

To detect memory growth before it causes an OOM kill, publish the Node.js heap usage alongside your heartbeat metric. For example:

```
setInterval(async () => {
  const mem = process.memoryUsage()
  try {
    await cw.send(new PutMetricDataCommand({
      Namespace: 'WickrIO/Bots',
      MetricData: [
        {
          MetricName: 'Heartbeat',
          Dimensions: [{ Name: 'BotName', Value: BOT_NAME }],
          Value: 1, Unit: 'Count'
        },
        {
          MetricName: 'HeapUsedMB',
          Dimensions: [{ Name: 'BotName', Value: BOT_NAME }],
          Value: Math.round(mem.heapUsed / 1048576),
          Unit: 'Megabytes'
        }
      ]
    }))
  } catch (e) { console.error('Metric publish failed:', e.message) }
}, 60000)
```

Create a CloudWatch alarm that fires when heap usage exceeds 80% of the container's memory limit. For example, if the container has a 512 MB memory limit (`--memory=512m`), set the threshold to 410 MB. Adjust the threshold to 80% of your configured limit.

```
MemoryAlarm:
  Type: AWS::CloudWatch::Alarm
  Properties:
    AlarmName: !Sub "${BotName}-HighMemory"
    Namespace: WickrIO/Bots
    MetricName: HeapUsedMB
    Dimensions:
      - Name: BotName
        Value: !Ref BotName
    Statistic: Maximum
    Period: 300
    EvaluationPeriods: 3
    Threshold: 410
    ComparisonOperator: GreaterThanThreshold
    AlarmActions:
      - !Ref AlertTopic
```

Remediation

- **Immediate:** Restart the container with `docker restart container-name`. This clears the Node.js heap without losing the WickrIO registration database.
- **Preventive:** Set a Docker memory limit with `--memory=512m` when starting the container to prevent a leaking bot from consuming all host memory.
- **Automated:** Schedule a weekly container restart using `cron` during a low-traffic maintenance window.
- **Root cause:** If heap usage grows steadily, the bot code likely has a memory leak. Common causes include unbounded session caches, event listeners that are never removed, and large message payloads stored in memory. Use the Node.js `--inspect` flag and Chrome DevTools to take heap snapshots and identify the leak.

Required IAM permissions

The Amazon EC2 instance profile for the bot host requires the following permissions for Amazon CloudWatch monitoring:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": "cloudwatch:PutMetricData",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:/wickr/bots/*",
                "arn:aws:logs:*:*:log-group:/wickr/bots/*:*"
            ]
        }
    ]
}
```

Collect bot logs

A Wickr IO bot produces logs at multiple levels. Understanding where each log is stored helps you collect the right information when troubleshooting.

Log	Path inside container	Contents
Docker container log	Access with `docker logs`	WickrIO console output and container startup messages. Bot integrations do not write to standard output, so `docker logs` does not contain application-level bot logs.
WickrIO server log	`/opt/WickrIO/logs/WickrIOSvr.log`	Server process events, client registration, and connection status. Check this log when the container is running but the bot is not connecting.

Log	Path inside container	Contents
Integration log (WickrLogger)	`/opt/WickrIO/clients/ bot-name/integration/ bot-name/ logs/log.output`	Application-level logs from your bot code, written by the WickrLogger library. This is the most useful log for debugging bot logic issues.

To view the integration log from outside the container:

```
sudo docker exec container-name \
  cat /opt/WickrIO/clients/bot-name/integration/bot-name/logs/log.output
```

To view the WickrIO server log:

```
sudo docker exec container-name cat /opt/WickrIO/logs/WickrIOSvr.log
```

View Docker container logs

```
# View recent logs
sudo docker logs --tail 100 container-name

# Follow logs in real time
sudo docker logs -f container-name
```

Configure integration log settings

The WickrLogger library reads log settings from the `processes.json` file in your bot's integration directory. Add a `log_tokens` section to configure the log level, maximum file size, and maximum number of rotated log files:

```
{
  "apps": [{
    "name": "bot-name",
    "script": "src/index.js",
```

```
    "env": {
      "tokens": { },
      "log_tokens": {
        "LOG_LEVEL": "debug",
        "LOG_FILE_SIZE": "10m",
        "LOG_MAX_FILES": "5"
      }
    }
  }]
}
```

The defaults are `info` level, 10 MB maximum file size, and 5 rotated files. Set `LOG_LEVEL` to `debug` temporarily when troubleshooting, then revert to `info` for normal operation. For more information, see Logging API in the *Wickr IO Bot Guide*.

Note

Always revert `LOG_LEVEL` to `info` after troubleshooting. Running at `debug` level in production generates excessive log volume and, on older bot versions, may log sensitive values such as authentication challenge tokens.

Note

When investigating bot crashes, check both the integration log and the Docker container log (`docker logs` *`container-name`*). Crash traces from unhandled exceptions and segmentation faults are written to the container log.

Configure Docker log rotation

Important

The default Docker log driver (`json-file`) does not rotate logs. The log file grows without limit until it fills the host's disk, which can cause the bot and other processes on the host to fail. Always configure log rotation for production bots.

Set the `max-size` and `max-file` options when you start the container. For example:

```
sudo docker run -it \
  --restart unless-stopped \
  --log-opt max-size=50m \
  --log-opt max-file=5 \
  other-options \
  image-name
```

This configuration keeps up to 5 log files of 50 MB each (250 MB total). To set this as the default for all containers on the host, add the following to `/etc/docker/daemon.json` and restart Docker:

```
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "50m",
    "max-file": "5"
  }
}
```

Forward logs to Amazon CloudWatch Logs

For production bots, you can forward container-level logs to Amazon CloudWatch Logs using the Docker `awslogs` log driver. This captures WickrIO server output and container startup messages, which is useful for detecting container crashes and restart loops. To forward bot application logs to Amazon CloudWatch Logs, use the CloudWatch agent on the host to tail the volume-mounted integration log file, or publish metrics directly from your bot code using the heartbeat approach described in the section called “Add a heartbeat metric (recommended)”.

```
sudo docker run -it \
  --restart unless-stopped \
  --log-driver=awslogs \
  --log-opt awslogs-region=us-east-1 \
  --log-opt awslogs-group=/wickr/bots/bot-name \
  --log-opt awslogs-create-group=true \
  other-options \
  image-name
```

The Amazon EC2 instance profile must include the `logs:CreateLogGroup`, `logs:CreateLogStream`, and `logs:PutLogEvents` permissions.

> **Note**
>
> Complete the initial bot setup (the `add` and `start` commands in the WickrIO console) using the default `json-file` driver first. After the bot is running, recreate the container with the `awslogs` driver. Use `docker restart` for subsequent restarts to preserve the registration database.

Archive logs to Amazon S3

For compliance or long-term retention, export Amazon CloudWatch Logs data to Amazon S3. You can configure automatic export using a Amazon CloudWatch Logs subscription filter with Amazon Data Firehose, or perform one-time exports from the Amazon CloudWatch console. For more information, see Export log data to Amazon S3 in the *Amazon CloudWatch Logs User Guide*.

Set a retention policy on the Amazon CloudWatch Logs log group to control costs. For most bot deployments, 30 to 90 days of log retention in Amazon CloudWatch Logs is sufficient for troubleshooting, with older logs archived to Amazon S3.

Common issues

Exited Docker container

Once the Wickr IO container is set up, it is designed to run indefinitely unless stopped manually. However, unexpected events may occur that could cause your running Docker container to exit.

Solution

- Run docker restart (container-name) to get your container back up.
- Look at Wickr IO client logs to identify the reason for crash. For more information, see Wickr IO clients logging.
- Check your host's disk space usage to ensure your host machine has enough disk space to support the clients running within the Wickr IO Docker container.

Exited Docker container

Once you have a running bot client, you should be able to interact(depends on the integration type) with it on Wickr client application. If the bot is not responding as expected see the solution below to debug the issue.

Solution

- Ensure the Wickr IO client is in `Running` state on the Wickr IO docker CLI. You can list all clients and their state using the `list` command.
- If the client is running, then see the integration logs for any errors. For more information, see Wickr IO clients logging.

Too many bot devices

Bot clients do not automatically remove old devices upon start-up. If you frequently move a bot between different machines, you may reach the limit of 50 devices. Each time you move the bot to a new device or recreate it, a new device entry is created. Once you reach this limit of 50 devices, the bot may stop functioning properly.

Solution

Starting from version 5.81, there are multiple ways to clear out old bot devices. To do this, you need to modify the bot's `.ini` file. This is a one-time setting that the bot client will use to eliminate old devices. The setting will specify the number of seconds a bot device has remained logged in. If a device's last login exceeds this specified duration, it will be suspended. For example:

```
[oneshot]
suspenddeviceafterseconds=120
```

In the example, you can set the value to 120 seconds (or two minutes). This setting will NOT affect the current device but will suspend all devices that have not logged in within the specified time frame. The `.ini` file can be found in the client directory associated with the bot client. For example:

`/opt/WickrIO/clients/[botname]/WickrIO[botname].ini`

Restart the bot after you have added this to the `.ini` file.

Wickr files are not received or transmitted

If the Wickr IO integration you are using requires sending or receiving files but you're experiencing issues with either, it may be due to an inaccurate date and/or time setting on your system. Wickr's system relies on the host date and time to transfer encrypted files successfully. If these setting are incorrect, file transfers will fail.

Solution

For accuracy, it's recommended to install and run an NTP (Network Time Protocol) service on your host system.

Network timeouts

If your bot is failing to interact with the backend servers and the logs indicate time errors, there is a way to increase the timeout for network requests. While this may not resolve the underlying problem, it can help rule out timeout as a cause of connection issues.

Solution

To adjust the network timeouts, you'll need to modify the `.ini` file for the bot client. The `/opt/WickrIO/clients/WickrIO.ini` is the appropriate file.

Add the following below to the `.ini` file:

```
[networksetup]
requesttimeout=[some value in seconds]
userrequesttimeout=[some value in seconds]
```

The `.ini` file is located in the client directory associated with the bot client, for example:

`/opt/WickrIO/clients/[botname]/WickrIO[botname].ini`

Restart the bot after you have added this to the `.ini` file.

Bot doesn't respond to messages

Possible causes:

- The bot process has crashed inside the container.
- The bot lost its connection to the Wickr network.
- The container is running but the integration was not started.

Resolution

1. Check container status: `sudo docker ps -a --filter name=`*`container-name`*
2. Check bot logs: `sudo docker logs --tail 50` *`container-name`*
3. Verify the bot shows as online in the Wickr Admin Console under **Bots**.
4. Restart the container: `sudo docker restart` *`container-name`*

Failed to create connection to host

The bot container logs show `Failed to create connection to host` when attempting to reach the Wickr messaging gateway on port 443. The bot cannot connect to the Wickr service and will not come online.

Possible causes:

- A firewall, security group, or network ACL is blocking outbound TCP port 443 from the bot host.
- A proxy or web filter is intercepting TLS connections to Wickr domains.
- DNS resolution is failing for the Wickr messaging gateway domain.
- The bot is using the wrong Docker image for its deployment type (for example, the commercial image in GovCloud or vice versa).

Resolution

1. Verify that the host can resolve and reach the Wickr messaging gateway. Run the following commands from the bot host (not from inside the container):

```
# For commercial deployments:
nslookup gw-pro-prod.wickr.com
curl -v https://gw-pro-prod.wickr.com

# For GovCloud deployments:
nslookup api.messaging.wickr.us-gov-west-1.amazonaws.com
curl -v https://api.messaging.wickr.us-gov-west-1.amazonaws.com
```

   If `nslookup` fails, the host has a DNS resolution issue. If `curl` fails with a connection timeout or reset, a firewall or security group is blocking the traffic.

2. Verify that the Amazon EC2 security group allows outbound TCP port 443. In the Amazon EC2 console, select the instance, choose the **Security** tab, and review the outbound rules for the attached security group.
3. Allowlist the required Wickr domains for your AWS Region. The bot host must be able to reach the messaging gateway domain on TCP port 443. For the full list of domains and IP addresses by Region, see Ports and domains to allow list for your Wickr network.
4. If the host is behind a corporate proxy or web filter, verify that the proxy is not performing TLS inspection on Wickr traffic. The Wickr IO container requires a direct TLS connection to the messaging gateway. Configure a proxy bypass for the Wickr domains listed in the allowlist.
5. Verify you are using the correct Docker image for your deployment type. Commercial deployments must use `wickrio/bot-cloud:latest`. GovCloud deployments must use `wickrio/bot-cloud-govcloud:latest`. Using the wrong image causes the container to connect to the wrong messaging gateway, which results in a connection failure.

Container starts but bot process fails

Possible causes:

- Missing Node.js dependencies.
- Incorrect bot credentials.
- The bot user was removed from the Wickr network.

Resolution

1. Check the container logs for error messages.
2. Verify the bot user exists in the Wickr Admin Console.
3. Re-run `npm install` inside the container to restore dependencies.

Bot was working but stopped after container recreation

If you removed and recreated the container (using `docker rm` followed by `docker run`) instead of restarting it (using `docker restart`), the WickrIO server's registration database was lost. The container starts, but the WickrIO server does not recognize any registered bot clients.

Symptoms:

- The WickrIO server log shows `WickrIO Client Server configured` but no connection attempts.
- The bot process starts but hangs at `Checking for client connection` indefinitely.
- In interactive mode, the console shows `There are no clients currently configured!`
- In detached mode (`-d`), the WickrIO server may crash with a segmentation fault (`Segmentation fault (core dumped)` in `docker logs`) and the container enters a restart loop. This occurs because the server attempts to auto-start registered clients in `-notty` mode, but the registration database is empty.

Resolution

1. Attach to the container: `sudo docker attach` *`container-name`*
2. Re-register the bot using the `add` command. If auto-login was previously configured and the client data volume is intact, the bot uses the existing auto-login keys.
3. Start the bot with `start 0`.
4. Detach from the container with `Ctrl+P`, `Ctrl+Q`.

To prevent this issue, use `docker restart` instead of `docker rm` and `docker run`. If you must recreate the container (for example, to change the log driver), plan for the re-registration step.

> **Important**
>
> If the container is running in detached mode (`-d`) with `--restart` enabled and the registration database is lost, the WickrIO server repeatedly attempts to re-register on each restart. This can exhaust the Wickr service's registration rate limit, which locks the bot username for 24 hours. Stop the container immediately with `docker stop` *`container-name`* if you see `Segmentation fault` or `Begin register existing user context` repeating in `docker logs`.

Bot is online but CloudWatch alarms are not working

Possible causes:

- The Amazon EC2 instance profile is missing `cloudwatch:PutMetricData` permission.
- The bot is publishing to a different AWS Region than the alarm.

Resolution

1. Verify the instance profile includes `cloudwatch:PutMetricData`.
2. Confirm the `AWS_REGION` environment variable matches the Region where your alarms are configured.
3. Check the CloudWatch console under **Custom namespaces** for the `WickrIO/Bots` namespace to verify metrics are arriving.

Bot integration does not start after container restart

The container is running and the WickrIO server shows the bot as `Running`, but the bot does not respond to messages and no integration log is produced.

Cause: When the container restarts in detached mode, the WickrIO server starts the Wickr client process but does not automatically launch the Node.js integration. The integration must be started separately.

Resolution

1. Start the integration manually inside the container:

```
sudo docker exec -d container-name bash -c \
  'source /usr/local/nvm/nvm.sh && nvm use 20 >/dev/null && \
   cd /opt/WickrIO/clients/bot-name/integration/bot-name && \
   node src/index.js > /tmp/bot.log 2>&1'
```

2. Verify the integration is running by checking for the Node.js process and the integration log:

```
sudo docker exec container-name pgrep -a node
sudo docker exec container-name tail /tmp/bot.log
```

> **Note**
>
> The `name` field in your bot's `package.json` must match the registered bot username. The WickrIO SDK uses this value to determine the IPC socket path for communicating with the

Wickr client process. If the names do not match, the integration starts but cannot connect to the client.

Integration import fails with "install shell file does not exist"

When importing a custom integration during the `add` command, the WickrIO console reports `install shell file does not exist for this software!` and the import fails.

Cause: The `software.tar.gz` file does not contain an `install.sh` script. The WickrIO import process requires this script to install the integration dependencies.

Resolution: Add an `install.sh` script to the root of your `software.tar.gz` archive:

```
#!/bin/bash
cd "$(dirname "$0")"
npm install --production
```

Rebuild the archive with the script included:

```
tar czf software.tar.gz src/ package.json install.sh
```

Upgrading bots

The WickrIO bots do not automatically upgrade, like a Wickr client does. You will need to periodically and manually upgrade the bots to make sure you have the latest software.

Solution

The following are the steps you will perform to upgrade a WickrIO docker image and the associated bots:

1. Pause the bots that are running on the WickrIO docker container. This is a safety precaution to make sure there are no database corruption problems when you upgrade the container. You will log into the WickrIO CLI and perform the 'pause' command for each bot you have running. After pausing the bots, exit out of the WickrIO CLI (using the <ctrl>p <ctrl>q keyboard sequence).
2. Next, you will kill the current docker container. The following commands will kill the running container and remove it. Notice we named our bot 'MyBotName', you may have named it

differently. If you ran the docker container with the appropriate "-v" option your bot data will be safe. The following is an example command:

```
docker kill MyBotName
docker rm -f MyBotName
```

3. Next, you will start the docker container with the new version and use the same options you used to start the old version. The following is an example command:

```
docker run -v /opt/MyBotName:/opt/WickrIO -p 4001:4001 -d --restart=always --
name="MyBotName" -ti wickr/bot-cloud:latest
```

4. Next, you will upgrade the bots running on this container. You will need to do that for each of the bots on this container. Log onto the WickrIO CLI, and perform the 'list' command, which will show you the list of bots and identify which ones need to be upgraded. Depending on how old your version is, it is likely all of your bots will require upgrading.
5. Next, for each bot perform the "upgrade" command. It is possible that the configuration tokens associated with the bot integration has changed, if so you will be prompted for the new values.
6. After ensuring all the bots have been upgraded, start each bot. When done starting you can perform the "list" command to verify each bot is running. Once verified, you can exit out of the WickrIO CLI.

The following is sample output from upgrading an old WickrIO bot running version v4.64.9.3, with a wickrio_web_interface bot. The version being upgraded to is the 5.116.18.01 version/tag.

```
Enter command:list
Current list of clients:
 client[0] old-test-bot, State=Running, Integration=wickrio_web_interface
Enter command:version
version: v4.64.9.3
Enter command:pause 0
Do you really want to pause the client with the name old-test-bot (default: yes):yes
Enter command:list
Current list of clients:
 client[0] old-test-bot, State=Paused, Integration=wickrio_web_interface
Enter command:read escape sequence
ubuntu@mybothost:~$
docker kill VeryOldBot
ubuntu@mybothost:~$
docker rm -f VeryOldBot
```

```
ubuntu@mybothost:~$
docker run -v /opt/VeryOldBot/WickrIO:/opt/WickrIO -p 4444:4444 -d --restart=always --
name="VeryOldBot" -ti wickr/bot-cloud:latest
Unable to find image 'wickr/bot-cloud:latest' locally
5.116.18.01: Pulling from wickr/bot-cloud
4bbfd2c87b75: Already exists
d2e110be24e1: Already exists
889a7173dcfe: Already exists
20ca454721b1: Already exists
02c4b05cb492: Already exists
1c731f8c023d: Already exists
fcd34fc70cfe: Already exists
b1fceb668295: Already exists
68307b81514b: Pull complete
b9f674e60307: Pull complete
e4fd99322c5a: Pull complete
3fdf8c473cd1: Pull complete
9dd71d5d554c: Pull complete
225a5f4a7590: Pull complete
Digest: sha256:05f7d0df8a488f7e325845423d13ebda95b1967b7b00203fe2425ae35e43480f
Status: Downloaded newer image for wickr/bot-cloud:latest
71bb0c85bc4c4c92db604136ad9f64e62d9769debcfccb656794037b4dc06d44
ubuntu@mybothost:~$ docker attach VeryOldBot

Updating integration software version numbers
Updating integration software version numbers
Searching NPM registry
Searching NPM registry

Welcome to the WickrIO Console program (v5.116.18.01)

Current list of clients:
#  Name          Status  Integration            Version  Events  Misc
=====================================================================
0  old-test-bot  Paused  wickrio_web_interface  1.1.2    2       Needs Upgrade!
Enter command:
Enter command:upgrade 0
Upgrading from version 1.1.2 to version 5.82.2
Okay to proceed? (default: yes):yes
Copying wickrio_web_interface software

Upgrading wickrio_web_interface software
Installing wickrio_web_interface software
Installing
```

```
Installing
Begin configuration of wickrio_web_interface software for old-test-bot
Now using node v12.20.2 (npm v6.14.11)
adminsOptional=false
adminsOptional=false
Finished Configuring package!
Finished Configuring forever!
Finished Configuring!
Enter command:list
Current list of clients:
#  Name          Status  Integration            Version  Events  Misc
=============================================================================
0  old-test-bot  Paused  wickrio_web_interface  5.82.2   2
Enter command:start 0
Preparing to start the client with the name old-test-bot
Do you really want to start the client with the name old-test-bot:yes
Enter command:list
Current list of clients:
#  Name          Status   Integration            Version  Events  Misc
=============================================================================
0  old-test-bot  Running  wickrio_web_interface  5.82.2   3
Enter command:
```

Improve bot resilience

A Wickr IO bot running on a single Amazon EC2 instance is a single point of failure. If the instance fails, the bot is offline until you manually intervene. The following practices improve bot availability for production deployments.

Persist container state across restarts

The Wickr IO container stores critical state in two locations that you must mount as Docker volumes to persist across container restarts:

- **Client data** (`/opt/WickrIO/clients/`*`bot-name`*`/client/`) – Contains the bot's encryption keys, auto-login credentials, and message database.
- **Integration code** (`/opt/WickrIO/clients/`*`bot-name`*`/integration/`) – Contains your bot's source code, `package.json`, and configuration files.

> **Important**
>
> The WickrIO server process maintains a registration database (`wickrbot.database.sqlite`) inside the container at `/opt/WickrIO/`. This database records which bot clients are registered. If you recreate the container (by running `docker rm` followed by `docker run`), this database is lost and the bot must be re-registered through the interactive console using the `add` command. To avoid this, use `docker restart` instead of removing and recreating the container.

The following example shows the recommended volume mounts:

```
sudo docker run -it \
  --name bot-name \
  --restart unless-stopped \
  -e PRODUCT_TYPE=govcloud \
  -v /opt/wickr-bot:/opt/WickrIO/clients/bot-name/integration \
  -v /opt/wickr-bot-client:/opt/WickrIO/clients/bot-name/client \
  image-name
```

Enable automatic instance recovery

Amazon EC2 simplified automatic recovery is enabled by default on supported instance types. If the underlying hardware fails, AWS automatically migrates the instance to healthy hardware. The instance retains its ID, IP addresses, and attached EBS volumes. For more information, see Automatic instance recovery in the *Amazon EC2 User Guide*.

Automate bot recovery with health checks

The Docker `--restart unless-stopped` policy restarts the container when it exits, but it does not detect cases where the container is running while the bot process inside it has stopped responding. To detect and recover from these silent failures, add a Docker health check.

Create a health check script in your bot's integration directory that verifies the bot process is running and producing logs. For example:

```
#!/bin/bash
# healthcheck.sh — exit 0 = healthy, exit 1 = unhealthy
```

```
# Check that the Node.js bot process is running
pgrep -f "node entry-point" > /dev/null || exit 1

# Check that the bot has produced a log entry in the last 5 minutes
LOG="/opt/WickrIO/clients/bot-name/integration/bot-name/logs/log.output"
if [ -f "$LOG" ]; then
  LAST_MOD=$(stat -c %Y "$LOG" 2>/dev/null || stat -f %m "$LOG" 2>/dev/null)
  NOW=$(date +%s)
  DIFF=$(( NOW - LAST_MOD ))
  [ "$DIFF" -gt 300 ] && exit 1
fi

exit 0
```

Start the container with the health check configured. For example:

```
sudo docker run -it \
  --restart unless-stopped \
  --health-cmd="/opt/WickrIO/clients/bot-name/integration/healthcheck.sh" \
  --health-interval=60s \
  --health-retries=3 \
  --health-start-period=120s \
  other-options \
  image-name
```

The `--health-start-period` gives the bot time to authenticate with the Wickr network before health checks begin. After the start period, Docker runs the health check every 60 seconds. If 3 consecutive checks fail, Docker marks the container as `unhealthy`.

Note

The Docker `--restart` policy does not automatically restart unhealthy containers. Add a watchdog script on the host to trigger a restart:

```
#!/bin/bash
# /opt/wickr-bot/watchdog.sh — run via cron every 2 minutes
CONTAINER="container-name"
```

```
HEALTH=$(docker inspect --format='{{.State.Health.Status}}' "$CONTAINER" 2>/dev/null)

if [ "$HEALTH" = "unhealthy" ]; then
  logger "Wickr bot $CONTAINER is unhealthy — restarting"
  docker restart "$CONTAINER"
fi
```

```
*/2 * * * * /opt/wickr-bot/watchdog.sh
```

Using `docker restart` preserves the WickrIO registration database and container logs.

Use an Auto Scaling group

An Auto Scaling group with a minimum, maximum, and desired capacity of 1 automatically replaces the instance if it fails a health check. Store your bot's client data and integration code on a separate Amazon EBS volume or Amazon EFS file system so the replacement instance can mount the same data and resume without re-registering the bot. For more information, see the Amazon EC2 Auto Scaling User Guide.

Use a container orchestrator

If your organization already uses a container orchestrator such as Amazon ECS, Amazon EKS, or a lightweight Kubernetes distribution, you can run the Wickr IO container as a managed task or pod with built-in health checks, automatic restarts, and log collection. This approach is recommended only if you already have orchestration infrastructure in place.

Release Notes

AWS Wickr bots and integrations are regularly updated with new features, improvements, and bug fixes. This chapter provides information about the changes in each release.

Version 6.66.02.01 - Release Date: 06/18/2026

The 6.66.02.01 release is a maintenance release focused on runtime modernization and security remediation.

Critical Changes

Runtime Modernization

- Upgrade to Node 24: The bot-cloud Docker image has been upgraded from Node 20 to Node 24. This is a runtime-only change no integration code changes are required. Existing bots will continue to function after upgrading to this image.

Security Remediation

- Fixed known CVEs (Common Vulnerabilities and Exposures)

Version 6.60.05.78 - Release Date: 01/20/2026

The 6.60.05.78 release is a bug fix release and focused on the following major changes:

Bug Fix: Enhanced authorization validation in bot's admin command handling

Admin command handling was found to have limited authorization validation. Changes were made to prevent unauthorized privilege escalation.

Version 6.60 - Release Date: 12/08/2025

AWS Wickr Bots 6.60 is a maintenance release focused on platform modernization, security compliance, and infrastructure improvements. This release includes critical updates to support Ubuntu 24, Qt 6, and FIPS compliance standards.

Critical Changes

Platform Modernization

- Ubuntu 24 & Qt 6 Upgrade: Major platform modernization for better performance
  - Upgraded to Ubuntu 24 LTS base operating system
  - Migrated to Qt 6 framework
  - Enhanced stability and long-term support

Security & Compliance

- FIPS Compliance Support: Added federal security standards compliance
  - Support for Federal Information Processing Standards (FIPS) 140-2/140-3
  - Enables deployment in federal and regulated environments requiring FIPS compliance
  - Cryptographic operations can be configured for FIPS mode

- Security Vulnerability Remediation:
  - Fixed known CVEs (Common Vulnerabilities and Exposures)
  - Updated dependencies to address security issues

Data Standards

- JSON Data Compliance: Fixed compliance issues with required timestamps and sender/receiver information
  - Ensures all message payloads include required timestamp fields
  - Corrected sender and receiver information formatting
  - Improved data consistency across JSON outputs

Network Connectivity

- SOCKS5 Proxy Support: Enhanced network connectivity options
  - Added support for SOCKS5 proxy protocol
  - Improved connectivity in restricted or enterprise network environments
  - Enables deployment behind corporate proxies

Infrastructure Improvements

Docker Architecture

- Reorganized container structure for improved maintainability
- Unified FIPS dependency management across containers

Build System

- Modernized build processes for improved reliability
- Updated Python dependencies to current versions

Performance Optimizations

- Optimized memory usage during bot operations
- Improved database query performance

Version 6.48 - Release Date: 04/14/2025

The 6.48.04.11 release of the bot-cloud and bot-dataretention-govcloud images are focused on the changes described below. The 6.48.04.01 bot-compliance-cloud and bot-dataretention-govcloud images include the CloudWatch metrics bug fix described below.

Upgrade to Node 20

The bot-cloud and bot-dataretention-govcloud docker images have been upgraded from Node 16 to Node 20. Prior to installing version 6.48 (and above), complete the steps in the section called “Version 6.48 announcement” to update your integrations and avoid disruption.

Deprecate old integrations

Some of the older and non-supported integrations have been removed from the docker images. For more information, see the section called “Version 6.48 announcement”.

WickrIO Addon to use ZeroMq

The WickrIO Addon is changed to use ZeroMq for all interactions between integrations and WickrIO client. This change will make the bot APIs asynchronous. For more information, see the section called “Version 6.48 announcement”.

Bug Fix: Data Retention Bot failing to publish CloudWatch metrics

Fixed issue where the Data Retention Bot was not publishing metrics to CloudWatch due to a missing parameter. This fix will ensure that all the required metrics are published successfully to CloudWatch - if users have CloudWatch configured.

Version 6.36.20.02 - Hotfix

The 6.36.20.02 hotfix release of the bot-enterprise image and the 6.36.09.01 hotfix release of Data Retention bot images are focused on the fix described below.

Bug Fix: The Edit message, delete message and reactions are not being captured

The Compliance bot and Data Retention are not processing a few types of control messages:

- Edit Messages: 9000
- Edit Reaction message: 9100
- Delete message: 4011

This hotfix will allow the compliance bot and Data Retention bot to process the above mentioned control messages and capture them in the receivedMessages files.

Version 6.36.13.01

The 6.36.13.01 release of the bot-cloud and bot-enterprise images are focused on the changes described below.

Add ability to suspend bot devices

Currently there is no way to suspend old bot devices. These changes will by default only allot 10 devices for each bot. There are two environment variables that allow you to remove devices older than a specific number of minutes, or change the limit on the number of devices.

Better logging for clientConfig.json bad JSON

Using the clientConfig.json file to automate the start of bots does not generate logs when there is a failure to parse the JSON. This fix will output errors when failure to parse the JSON occurs.

Bug Fix: Sending messages to invalid users could hang bot

There have been situations where trying to send messages to invalid users will hang a bot. This has been easily reproducible using the broadcast bot when sending to a list of users in a file. This fix addresses this issue.

Version 6.34.05.01

The 6.34.05.01 release of the bot-cloud and bot-enterprise images are focused on the changes described below. The 6.34.02.01 bot-compliance-cloud images include the File Management and SDK changes described below.

Support File Management feature message formats

The AWS Wickr control message formats were changed to support the AWS Wickr File Management feature. The 6.34 versions of the WickrIO and WickrIO Compliance releases support this new message format. The Data Retention / Compliance bots will also download files that are referred to in these modified control messages. Details of these updated control messages will be detailed in the appropriate WickrIO documentation.

Bug Fix: Broadcast Bot security group selection failed

Some AWS Wickr clients have added spaces padding the security group selection which broke the broadcast bot processing of that string. Added coding in the broadcast bot to trim the padding off of the response string.

Bug Fix: Bug fixes inherited from lower layer SDK

Several bug fixes were made to the SDK that bots use to interact with the AWS Wickr servers for login and messaging services. Changes made to improve room membership for room conversations. Changes made to improve the AWS Wickr client record information more reliable. Removed some code associated with legacy message sends, which is not used any more.

Version 6.32.04.02

The 6.32.04.02 release is a bug fix release and focused on the following major changes:

Bug Fix: Airgap version contacting NPM Registry

Some code was found to still have references to the default NPM registry (registry.npmjs.org). Changes were made so the Airgap version does not reference any NPM registry.

Bug Fix: Read receipts not working for Broadcast Bot

Software was found to not be decoding read receipt API call responses correctly. Changes were made to fix this.

Bug Fix: Registration failures

A race condition was found which caused initial registrations to fail.

Bug Fix: Read receipts never time out

Changes were made to stop sending requests for read receipt status after one week for broadcasts.

Version 6.24.06.02

The 6.24.06.02 release is focused on the following major changes:

Bug Fix: Conversations not restored when creating new instance of bot

Found issue where AWS Wickr conversations were not being restored correctly for new instances of a bot. This issue would present itself if you created a new instance of a bot and then tried to send a message from the bot to a secure room or group conversation. The bot would not have restored the connection list and would not have a record of the conversation.

Bug Fix: Downloading files in multi-domain environments

Found issue where the downloading of files from clients in different domains was not working for bots. This change will make sure files are downloaded when a bot downloads a file from a AWS Wickr client from another federated domain.

Bug Fix: Handle files with long file names

When a bot receives a file with a long file name, approximately 255 characters, it adds some information to the filename which may make the file name larger than 255 characters. The bot would end up dropping the file in this case, due to operating system limitations. This fix will remove any characters at the end of the file name to keep the length under 255 characters.

Feature to send events to AWS Amazon SNS Topic

To improve the health capabilities of AWS Wickr bots we added the ability to send events generated on a bot to an AWS Amazon SNS Topic. This topic can be used to send events to an email address or any other endpoint that can subscribe to events pushed to the defined Amazon SNS Topic. To use this feature there are environment variables (to be defined in the WickrIO documentation) that identify the AWS Amazon SNS Topic.

Created new API to set avatar for the bot client

This new bot API allows bot developers to set the avatar associated with the bot client. Details of this API will be defined in the WickrIO documentation.

Version 6.18.19.02

The 6.18.19.02 release is focused on the following major changes:

Continue using AWS Amazon ECR to host Docker images

We are starting to transition hosting our AWS Wickr bot docker images on AWS Amazon ECR. These are the AWS Amazon ECR repositories used to host the AWS Wickr bot docker images:

public.ecr.aws/x3s2s6k3/wickrio/bot-cloud

public.ecr.aws/x3s2s6k3/wickrio/bot-enterprise

public.ecr.aws/x3s2s6k3/wickrio/bot-compliance-cloud

We will continue to host production docker images on DockerHub for the next couple of releases.

Two-way data retention support

The AWS Wickr bot clients will support running with federated networks that are running data retention. When sending messages to clients in federated data retention active the bot client will also send the message to the appropriate data retention bot(s).

Data retention bots support additional user information

The SAAS data retention bot (bot-compliance-cloud) will output additional information about the users associated with a message. This new information includes the network ID as well as room membership.

> **Note**
>
> Some of this information is not immediately available to the data retention bot. For the network ID the data retention bot will need to interact with the server to get the information. The network ID for a user may not be known by the data retention bot until the server responds with that information. For the room membership information the data retention bot must capture room membership control messages that contain the current room membership. The data retention bot cannot show a valid list of room conversation members unless it sees this information. In this case an indication of the membership validity will be included with the message information.

Version 6.16.19.01

The 6.16.19.01 release is focused on the following major changes:

Continue using AWS Amazon ECR to host Docker images

We are starting to transition hosting our AWS Wickr bot docker images on AWS Amazon ECR. These are the AWS Amazon ECR repositories used to host the AWS Wickr bot docker images:

```
public.ecr.aws/x3s2s6k3/wickrio/bot-cloud
```

```
public.ecr.aws/x3s2s6k3/wickrio/bot-enterprise
```

We will continue to host production docker images on DockerHub for the next couple of releases.

Update image from Ubuntu 18.04 to Ubuntu 20.04

The Ubuntu operating system version was upgraded to Ubuntu 20.04 in all AWS Wickr bot docker images.

Fix message send error issues

If the bot attempted to send a message to a user and that message failed to send the indication to the bot software was not indicated. The bot software would get into a frozen state and not able to proceed. This was seen with broadcasts to groups of users where a user account may be in a bad state where the bot would fail to send to it. The bot would be stuck and not able to function. This fix handles the error cases appropriately, and in the case of the broadcast bot the /report command will show the failed users.

Support for AWS Wickr Multi-Region

All AWS Wickr bot images will support bot clients created in other AWS Regions, for example ca-central-1.

Fix Broadcast bot not receiving messages from users

If the bot sends a message to a user, before the user has sent a message to the bot, the bot software will not process the message. The creation of the user record in the database was not attaching to get message indications if the user record was created by the bot sending the initial message for that user. This can happen, for example, when setting up a broadcast bot with administrators. The broadcast bot would start by sending a welcome message to all of the administrators. The user record was not created to receive the incoming message indications, so the broadcast bot would not respond. In those cases restarting the broadcast bot would fix the indication for those users. There may be other situations where this happens.

More user-friendly bot startup failure indications

If a bot fails to startup the logs indicate this using an error code. To help indicate the actual reason for the failure a failure string will also be output to the logs.

Fix issue where bot startups more than 5 attempts will stop trying to start

If a bot fails to start more than 5 times, normally the bot will not attempt to start again without some user intervention. In some cases, the bot service will keep trying to start the bot which could lead to the bot client being suspended or the bot docker image CPU usage elevating too high. The change will keep the bot service from retrying to start the bot. The bot CLI will also indicate that there was an error in the Misc column. The CLI will also prompt the user to make sure they want to start the bot, if it was stopped due to it not starting more than 5 times.

Update control message to indicate rooms with saved links and files

Certain control messages did not indicate the filevault information associated with saved links and files, specifically when new users were added to rooms.

Version 6.11.05.01

This version is an update for all bots with the move from the forever process manager to the WPM2 process manager as well as more specific updates to the broadcast bot, compliance bot, bot client, and wickrio-docs. The 6.11.05.01 release is focused on the following major changes:

Using AWS Amazon ECR to host Docker images

We are starting to transition hosting our AWS Wickr bot docker images on AWS Amazon ECR. These are the AWS Amazon ECR repositories used to host the AWS Wickr bot docker images:

```
public.ecr.aws/x3s2s6k3/wickrio/bot-cloud
public.ecr.aws/x3s2s6k3/wickrio/bot-enterprise
```

We will continue to host production docker images on DockerHub for the next couple of releases.

Move from Forever process manager to WPM2

This change affects all bots as they were all using the forever process manager. WPM2 is an internal project that can be found on NPM at: https://www.npmjs.com/package/wpm2

With WPM2 we rely on the use of PID files to keep track of the running process and the ability to kill the running process based on the PID (process ID) and WPM2 handles keeping the bot running

and ending the process gracefully when we go to pause or restart the bot. Here is an example of what the new package.json scripts look like:

```
"restart": "kill $(cat $(cat pidLocation.json)) && nohup wpm2 start --no-metrics ./
wpm.json >>wpm2.output 2>&1 & echo $! > $(cat pidLocation.json)",
"start": "nohup wpm2 start --no-metrics ./wpm.json >>wpm2.output 2>&1 & echo $! > $(cat
 pidLocation.json)",
"stop": "kill $(cat $(cat pidLocation.json))"
```

Performance improvement for large broadcast

Performance has been improved for large broadcasts in the broadcast bot. The pre-send preparation of the broadcast bot will see an obvious increase in performance for large broadcasts.

Updating the Support email in all the bots

In any bots and documentation that made reference to AWS Wickr support we were using the old support@wickr.com email. All these instances have been updated to the new support email: wickr-support@amazon.com.

Updated JSON timestamp

An ISO format timestamp has been added for message JSON. This was done in the compliance bot and all messages passed to bots and integrations.

Mutex Lock Enhancements

Mutex lock enhancements have been made to reduce the possibilities of database lockups.

Version 5.116.19.02

This version is a bug fix release.

Fix for Enterprise updated password not showing

This version of the enterprise docker image(s) will fix an issue where the changed password was not being shown if you run the "debug on" command and then the "debug off" command on the CLI. The debug state was not being set correctly to off when the "debug off" command was performed, which caused a problem processing the messages from the provisioning software. The debugging output was stopped but the internal state was not updated appropriately.

Version 5.116.18.01

This version is an update for the wickr/bot-enterprise and the wickr/bot-cloud Docker images. The 5.116.18.01 release is a patch release. Please see the Version 5.116.13.01 Release notes for details of the 5.116 version. This version includes the following changes:

Fix for Missing Rooms

A bug was detected when upgrading to the 5.112 or 5.112.13.01 releases of the WickrIO Bots, which made secure room and group conversations not visible to bots/integrations. These conversations were not deleted they were not being made visible to the bot APIs. This release fixes these issues.

Fix for upgrades from old bot versions

This issue was seen when an upgrade was performed from an old version of the WickrIO docker image, specifically 5.62.05.02. The issue was caused by a new database field not being added during the migration from the old version to the recent version (i.e. 5.112 or 5.116).

To identify if your bot is experiencing this issue, the log files will contain a failure message when ever the bot receives a message. The failure message will contain the following text and can be seen in the WickrIO<botname>.output file:

```
BULK INSERT MESSAGE: Failed, can't prepare insertQuery
```

Fix to address high CPU

Several customers have reported WickrIO bots randomly running at high CPU rates, and staying at that rate. We have reproduced the problem. This patch release contains a change that has shown to eliminate this problem. If you have seen this issue and want to confirm it is the same CPU issue please do the following:

- run the following command which will start a bash shell inside the docker image.

```
docker exec -ti <dockerimagename/id> bash
```

- run the "top" command which will show you the processes running inside the docker image. The process at the top of the list will have the highest CPU value.

- The process at the top of the list should be the "wickrio_bot" process, if it is the problem we are trying to solve.

> **Note**
>
> If you continue to have the CPU issue after installing this patch, please contact support.

Fix for SAAS Data Retention Network Transmit Failures

The 5.116.13.01 wickr/bot-cloud image would have transmit failure when operating in a SAAS Data Retention network. This version will fix that problem.

Version 5.116.13.01

This version is an update for the wickr/bot-enterprise and the wickr/bot-cloud Docker images. The 5.116.13.01 release is focused on the following two major changes:

Support for Mac M1 Host

Changes made to support running on Mac M1 machines. Older versions of the WickrIO Docker images will not run on the Mac M1 systems.

If you want to run the WickrIO Docker image(s) on a Mac M1 machine, you will have to start the Docker images with the —platform=linux/amd64 option.

We do not recommend running production bots on Mac M1 hardware at this point. Using Mac M1 systems to do development should be fine.

Support for Node 16

As of the 5.116.13.01 version, all of the Docker images and software are updated to use node version 16. Node 12 is not supported anymore and will NOT be part of the 5.116.13.01 Docker images. All of the bots and integrations shipped with the 5.116.13.01 Docker images have been updated to use Node 16. When you update your WickrIO Docker images to the 5.116.13.01 version, you will need to upgrade all bots and integrations to use the latest versions. Any bots that use Node 12 will not work when the 5.116.13.01 version is installed. The bot client will operate and show that it is running, but the bot/integration software will not be able to start up since they will

be looking for Node 12. The 5.116.13.01 WickrIO Docker images contain upgrades for all of the supplied bots.

Custom Bots/Integrations

This section applies to you if you have or are developing custom Node-js bots. This does not apply to anyone using the REST APIs of the wickrio_web_interface bot.

If you are creating custom bots/integrations you will need to use the latest 5.113 version of the wickrio-bot-api or the the latest 5.113 version of the wickrio_addon (which ever is appropriate for your custom bot).

Installation Process

We recommend that before you install the new 5.116.13.01 version of the appropriate Docker image that you pause all of the bots running on each Docker instance. Using the WickrIO CLI you should see your bot(s) in the "Paused" state.

```
Enter command:read escape sequence
ubuntu@ip-172-31-25-75:~$ docker attach bot_test_prod
Enter command:
Enter command:list
Current list of clients:
# Name Status Integration Version Events Misc
0 dev-test-bot Paused wickrio-compliance-bot 5.112.1 53
Enter command:
```

After you start the Docker image with the 5.116.13.01 version the CLI will show that your bot has a needed upgrade available:

```
Welcome to the WickrIO Console program (v5.116.13.01)

Current list of clients:
# Name Status Integration Version Events Misc
0 dev-test-bot Paused wickrio-compliance-bot 5.112.1 43 Needs Upgrade!
```

Please make sure you upgrade each of your bots to the latest version. The following is a sample of what upgrading a compliance bot would look like. The software will be upgraded and you will be prompted to enter the configuration information for the bot (previously configured values are shown in the parenthesis).

FAQ

Question: Will my bots be running if I don't want to upgrade?

Answer: Your bots will continue to run if you do not upgrade to the 5.116.13.01 version. However, we recommend upgrading to receive the latest security updates.

Question: If I run into problems with the 5.116.13.01 version can I downgrade my installation?

Answer: Yes you can. The process is very similar to how you upgraded to the 5.116.13.01 version. Instead you will do the following:

- pause the bots running on the Docker instance
- start the older Docker image
- use the CLI's 'upgrade' command to install the version of the bot software on the older Docker instance.
- start the bots