## Version History

<table>
<thead>
<tr>
<th>Version</th>
<th>Version Date</th>
<th>Authors</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>November, 2015</td>
<td>Alan Capps, James Lapic, and Scott Johnson</td>
<td>Original version</td>
</tr>
<tr>
<td>1.1</td>
<td>March, 2016</td>
<td>Alan Capps, James Lapic, and Scott Johnson</td>
<td>Corrections and minor updates.</td>
</tr>
</tbody>
</table>
| 2.0     | May, 2017     | Alan Capps and Scott Johnson | Summary of Changes:  
  - Expanded Summary to include three message-send scenarios, and describe difference between a message request and a messaging request.  
  - Expanded Authentication to cover single-user and multi-user scenarios.  
  - Created separate sections for each message-send scenario: single message, scheduled message, and message with attached image.  
  - Created separate section for status codes.  
  - Improved tables and examples. |
# Contents

1.0 Messaging API Summary 5  
1.1 Message (SMS) Versus Messaging (MMS) 5  
1.2 Sending and Receiving Text Messages 6  
2.0 Zipwhip Authentication 7  
2.1 Obtain a Session Key 7  
2.1.1 Example: Session key request 7  
2.1.2 Example: Session key response 8  
2.2 Single-User Versus Multi-User Authentication 8  
2.2.1 Single-user authentication 8  
2.2.2. Multi-user authentication 9  
3.0 Send a Text Message 10  
3.1 Send a Text Message to a Single Contact 10  
3.1.1 Example: Single-contact text message 10  
3.1.2 Table: Single-contact input parameters 11  
3.1.3 Example: Single-contact success response 11  
3.1.4 Table: Single-contact success output parameters 12  
3.1.5 Example: Single-contact failure response 12  
3.1.6 Table: Single-contact failure output parameters 13  
3.2 Status Codes 13  
4.0 Receive a Text Message 14  
4.1 Web Hooks: Retry Logic 14  
4.1.1 Single Message/Intermittent Message delivery failure 14  
4.1.2 All message delivery failure 14  
4.2 Web Hooks: Message Security 15  
4.3 Access Control Lists (ACL) 15  
4.4 Web Hooks Events 15  
4.4.1 Example: Web Hooks payload 16  
4.5 Install Web Hooks 16  
4.5.1 Example: Web Hooks install (add) request 17  
4.5.2 Table: Web Hooks installation parameters 17  
4.5.3 Example: Web Hooks installation response 18  
4.6 List Web Hooks 18

© Copyright 2017 Zipwhip, Inc. All rights reserved
6.4.9 Example: File too big failure response: 31

7.0 Status Code Summary 32

7.1 Table: “Message Not Sent” Status Codes 32
7.2 Table: “Message Progress” Status Codes 33
1.0 Messaging API Summary

This document describes how to use the Zipwhip Messaging API to perform two primary functions: to send text messages and to receive text messages. The document includes several message-send scenarios: how to send a text message, send a scheduled text message, and send a message with an attached image. Each scenario includes an example message, example responses, and input/output parameters. Also included are lists of status and error codes, and a description of how to use Web Hooks to receive text messages.

The intended audience for this document is software developers who are responsible for using Zipwhip’s API to add text-messaging functionality to their product, system, or network.

1.1 Message (SMS) Versus Messaging (MMS)

The Zipwhip Messaging API supports two types of message requests: message and messaging. Message allows you to send text-only messages and scheduled text messages. Messaging allows you to send text-only messages and text messages with an attached image. However, you cannot use messaging to send a scheduled text message. This means that if you want to set up your system to match all three scenarios, then you have to use both message and messaging.

Also, messaging does not allow you to automatically add your signature to outgoing text messages. If you want to automatically include a signature, then you must use the message request. If you use the messaging request, then you must manually add your signature to the outgoing text message.

<table>
<thead>
<tr>
<th>Zipwhip API Send Requests: Messaging/Message</th>
<th>Message Request</th>
<th>Messaging Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Request</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Send a text-only message</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Send a scheduled text-only message</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Send a text message with an image</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
1.2 Sending and Receiving Text Messages

The Zipwhip API gives you an engine to send and receive text messages. Zipwhip handles the send and receive functions, including authentication, routing to the appropriate carrier network, and actual delivery of the text message. To send messages, you need to develop and implement methods for adding the contact and composing the message body. The Zipwhip API processes the send request, and routes and delivers the message to the recipient’s mobile device.

To receive messages, you need to develop and implement methods to track real-time events (such as received message) and display messages that are received. Zipwhip uses Web Hooks to push state-changes to your account. Web Hooks allow Server-to-Server communication about new events without the requirement of a persistent connection. Web Hooks give you the ability to receive and track real-time events within the Zipwhip system.

This document includes Web Hook installation and operating instructions in Section 4 Receive A Text Message.
2.0 Zipwhip Authentication

Authentication is a primary element of using the Zipwhip API. Zipwhip's API relies on a session to authenticate API requests for our Messaging API. However, before you can acquire a session key, you must set up a Zipwhip account. Without an active Zipwhip account, you cannot get a session key.

To set up an account, please contact sales@zipwhip.com. If you already have an active account but cannot obtain a session key, then contact api@zipwhip.com for support.

When you set up and activate an account, remember that you must text enable every phone number for which you want a session key. When you activate your account, you can then obtain a session key for each text-enabled line.

2.1 Obtain a Session Key

Zipwhip's API relies on a session to authenticate API requests to our Messaging API. This is different from other messaging APIs that use an apiKey in conjunction with a secret. In Zipwhip's case, when a request is made with a session, then the system performs the action based on the static phone number associated with the provided session.

This session is a GUID representation of an authenticated user. You must obtain a session key for each line that you text enable. This means that if you have three lines that you want to use for texting, then you must text enable the three lines and obtain a separate session key for each line.

A Zipwhip session does not expire. To expire a session, you must use the user/logout API request. Because sessions do not expire, we recommend that you save the session created during the user login to a database. In the database, you must map the text-enabled phone number to the session.

We recommend that you do not hardcode this session into your code base. If an infinite loop occurs in your system, then it is much easier to delete a database row than it is to redeploy your code.

**Important!**

The session key is specific to each phone number. If you have three text-enabled lines, then you must make three separate session-key requests.

2.1.1 Example: Session key request

To obtain a session key, you perform a user/login web-call for the account:

$ curl -X POST https://api.zipwhip.com/user/login \
   -d username=2065551212 \
   -d password=mypassword
2.1.2 Example: Session key response

In this case, the response is the session key (highlighted) for the account.

```
{
  "success":true,
  "response":"3d0f1dde-aaff-4ce8-b61a-af212a860abc:123456789"
}
```

2.2 Single-User Versus Multi-User Authentication

The Zipwhip Messaging API supports both single-user and multi-user authentication. If you use single-user authentication, then all users are Administrators (Admin). There is a single tier of users.

If you use multi-user authentication, then at least one user is the Administrator and all other users are Operators. There are two tiers of users. Admins can start system sessions and change system settings. Admins also add Operators to the system. Operators can start a user-specific session, and then send and receive messages. Operators cannot change system settings.

2.2.1 Single-user authentication

Single-user authentication means that there is a single username and password assigned to the line. This username and password starts a session that has full administrative access. If more than one person want to use the line, then they must share the user name and password. However, whoever uses the username and password to start a session has full access.

The username for single-user logins must be the text-enabled phone number. For example, if you text-enable 425-555-1212, then the username you use for login requests must be 4255551212.

The password must have more than six (6) characters.

The following is an example of a single-user login request:

```
$ curl -X POST https://api.zipwhip.com/user/login \
  -d username=4255551212 \
  -d password=mypassword
```
2.2.2. Multi-user authentication

Multi-user authentication means that there are several tiers of authentication. Each text-enabled line can be assigned only a single phone number, but it can be assigned multiple usernames and multiple passwords. In a multi-user scenario, you can authenticate as:

- **The System**, which means that you use the phone number assigned to the line and a password to log in. The password is a string with a minimum of six alphanumeric characters. As the System, you have full administrative rights. However, because you log in with the phone number assigned to the line, there is no username associated with the session. You should log in as the System only to perform administrative tasks, update software, or troubleshoot system issues.

- **An Administrator** (Admin), which means that you use a username and password to log in. The Admin username is a combination of personal prefix and the phone number assigned to the line as the suffix: scott@4245551212. The password is a string with a minimum of six alphanumeric characters. Admins have the full administrative rights, the same rights as when you log in as the System. However, because you log in with an Admin username, a username is associated with the session.

- **An Operator**, which means that you use a username and password to log in. The Operator username is the same format as the Admin username, and the password must also be a minimum of six alphanumeric characters. Operators have limited rights: they can send and receive messages. The Operator's username is associated with the session.

When you start a session as the System or an Administrator, it is a “system” session, which means that the Admin can make changes to the system or send/receive messages. In multi-user scenarios, we do not recommend that you use system sessions for sending and receiving messages. Instead, use Operator sessions for sending and receiving messages.

When an Operator starts a session, it is a “user-specific” session that does not allow the Operator to make changes to the system. The Operator can only send and receive messages.

User-specific sessions do not expire. To end a user-specific session, developers must create a custom mechanism to end the Operator session. If you do not develop a method to end sessions, then the Admin must do one of the following after the Operator logs out:

- The Admin ends the session by logging out the Operator
- The Admin removes the Operator
- The Admin changes the Operator status to Inactive

The following is an example of an Operator login request:

```bash
$ curl -X POST https://api.zipwhip.com/user/login \
  -d username=scott@425551212 \
  -d password=scottpassword
```
3.0 Send a Text Message

In the Zipwhip API, there are two categories of messages:

- Zipwhip Originated (ZO) messages, which are outgoing text messages that are sent from the Zipwhip system
- Mobile Originated (MO) messages, which are consumer generated, incoming text messages that are received by the Zipwhip system

This section focuses on sending Zipwhip Originated messages. For more information about receiving Mobile Originated messages, see Section 4.0 Receive A Text Message.

The Zipwhip API supports three basic Zipwhip Originated (ZO) message-send scenarios:

- Single-contact text message
- Single-contact scheduled text message
- Single-contact text message with an attached image

3.1 Send a Text Message to a Single Contact

To send a ZO text message to a single contact (recipient), the message must include three parameters: session, contacts, and body. An error message is sent if your ZO message does not include these three elements. The message must include the body parameter, but this does not mean that there must be text (content) in the body; it means that you can send a blank message.

3.1.1 Example: Single-contact text message

The following is an example of a ZO text message sent to a single contact that includes the three required parameters and no optional parameters:

```
$ curl https://api.zipwhip.com/messaging/send \
   -d session=3d0f1dde-aaff-4ce8-b61a-af212a860abc:123456789 \n   -d --data-urlencode to='+18559479447' \n   -d --data-urlencode body='This is the message body from Zipwhip!'
```
3.1.2 Table: Single-contact input parameters

The table below includes all of the required and optional parameters that you can use when sending a ZO text message.

**Note:** The Zipwhip API does not support any additional input or output parameters that you may develop.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>The GUID representation of a logged-in user.</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>
| to             | The string representation of a phone number.  
• US Domestic numbers are full, 10-digit numbers.  
• All numbers must be in E.164 format. | String  | Yes      |
| body           | The message to be sent. The message body can be a maximum of 250 bytes. | String  | Yes      |

3.1.3 Example: Single-contact success response

The following is an example of the response to a ZO text message being sent successfully to a single contact. This example includes both required parameters (highlighted) and optional parameters.

```json
{
    "response": {
        "fingerprint": "4233621183",
        "root": "664880513175330816",
        "tokens": [
            {
                "to": 3460624702,
                "device": 304397802,
                "fingerprint": "4233621183",
                "message": "664880513175330816"
            }
        ]
    },
    "success": true
}
```

Technically, you can ignore the fingerprint and root elements of the response object. You can use the elements in the tokens array instead of the fingerprint and root elements. If the message has a single recipient, then there is a single token in the token array. If the message has multiple recipients, then there are multiple tokens in the array.
### 3.1.4 Table: Single-contact success output parameters

The table below includes required output parameters included in the response to a ZO text message that is sent successfully to a single contact.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>contact</td>
<td>The ContactId of the message recipient.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>device</td>
<td>UserId of the message sender.</td>
<td>Integer</td>
<td>Yes</td>
</tr>
<tr>
<td>fingerprint</td>
<td>The fingerprint is the ConversationId, which is an ID assigned based on the recipients that are included with the outgoing text message.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>message</td>
<td>The ID of the message that was just sent.</td>
<td>Integer</td>
<td>Yes</td>
</tr>
<tr>
<td>success</td>
<td>The success/failure of the web request</td>
<td>Boolean</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### 3.1.5 Example: Single-contact failure response

The following is an example of the response to a single-contact, ZO text message that was not sent successfully. This example includes all of the required parameters (highlighted) and optional parameters.

```json
{
    "response": {
        "code": -703,
        "desc": "Bad arguments"
    },
    "success": false
}
```
3.1.6 Table: Single-contact failure output parameters

The table below includes required and optional output parameters for a single-contact, ZO text message that was not sent successfully.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>The Zipwhip error code.</td>
<td>Integer</td>
<td>Yes</td>
</tr>
<tr>
<td>desc</td>
<td>Description of the error.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>success</td>
<td>The success/failure of the web request. True=success; False = failure.</td>
<td>Boolean</td>
<td>No</td>
</tr>
</tbody>
</table>

3.2 Status Codes

The Zipwhip API uses status codes to indicate the success, failure, or progress of an event. The success codes are assigned when message delivery succeeds to both the recipient and database. Failure codes are assigned when the message cannot be sent because of an internal or external problem. Progress messages are assigned when the message is in-process or queued. If you use Zipwhip’s API Web Hooks, then there are specific progress messages that allow you to track the message through each step of the delivery process.

For a complete list of Zipwhip status codes, see Section 7 Status Code Summary.
4.0 Receive a Text Message

Zipwhip uses Web Hooks to push state-changes to your account. Web Hooks allow Server-to-Server communication about new events, without requiring a persistent connection. Web Hooks give you the ability to receive and track real-time events within the Zipwhip system.

If you use Web Hooks, then you must install Web Hook events on each of your text-enabled lines. If you have a single text-enabled line, then you need to install Web Hooks on that line. If you have five text-enabled lines, then you need to install Web Hooks on all five lines. If you want to install all five Web Hook events on all five lines, then you must send a total of 25 install requests (five events x five lines = 25 requests).

Web Hooks are not required. However, without Web Hooks, you will need to use some mechanism to send, receive, read, and delete messages. The remainder of this section describes how to install and use Web Hooks to process message events.

4.1 Web Hooks: Retry Logic

The Zipwhip API uses Web Hooks to manage and execute the text-message retry processes. There are two processes: a retry process for single or intermittent delivery failure and a retry process when all messages fail to be delivered for one hour.

4.1.1 Single Message/Intermittent Message delivery failure

If there is a delivery failure for a single message or intermittent delivery failures, Zipwhip defines the message delivery as failed when one of the following conditions is met:

- If your web server responds with a status code that is not a 200-level code, then message delivery has failed
- If your server takes longer than seven (7) seconds to respond to a delivery, then message delivery has failed

When message delivery fails, the message is sent to the retry queue. Zipwhip uses an exponential back-off algorithm with a ten (10) minute ceiling for resending. The first retry occurs when the message has been in the queue for ten (10) seconds. Zipwhip continues to retry delivery for a maximum of 72 hours.

4.1.2 All message delivery failure

If all messages fail to be delivered for one (1) hour, then Zipwhip assumes that the host is down and stops trying to send messages. Zipwhip waits ten (10) minutes and then tries to send a single, relevant message. If the single message is delivered successfully, then Zipwhip queues the backlog of unsent messages and starts sending. If the single message is not delivered successfully, then Zipwhip continues sending a single message every 10 minutes. Zipwhip can continue to retry delivery for a maximum of 72 hours.
4.2 Web Hooks: Message Security

It is important that you set up Web Hooks in a manner that maintains security, including protecting the users identity and data.

- We recommend that the destination address is HTTPS
- Zipwhip supports TLS V1.0 and greater. Zipwhip does not support SSL V3.0 and older

4.3 Access Control Lists (ACL)

If you have strict firewall rules, or would prefer to lock down your Web Hook end-points to our IP space, use the ranges provided below.

69.46.44.0/24
208.69.95.64/26

4.4 Web Hooks Events

The Zipwhip API supports the following Web Hooks events. When implementing your specific application, you can choose the events for which you want to listen:

- Send
- Progress
- Receive
- Read
- Delete
4.4.1 Example: Web Hooks payload

The following is an example of the standard incoming Web Hooks payload.

```
POST /message/receive HTTP/1.1
Host: http://www.yoururl.com/zipwhip/api/receive
Content-Length: 581
Content-Type: application/json; charset=UTF-8

{
  "body":"Thanks for texting!",
  "bodySize":19,
  "visible":true,
  "hasAttachment":false,
  "dateRead":null,
  "bcc":null,
  "finalDestination":"4257772300",
  "messageType":"MO",
  "deleted":false,
  "statusCode":4,
  "id":634151298329219072,
  "scheduledDate":null,
  "fingerprint":"132131532",
  "messageTransport":9,
  "contactId":3382213402,
  "address":"ptn:/4257772222",
  "read":false,
  "dateCreated":"2015-08-19T16:53:45-07:00",
  "dateDeleted":null,
  "dateDelivered":null,
  "cc":null,
  "finalSource":"4257772222",
  "deviceId":299538202
}
```

4.5 Install Web Hooks

To process message events for a given text-enabled phone number, you must install Web Hook events on the given line. After you install a Web Hook on the given number, ZipWhip delivers the details of any message events that occur on that number to the specified URL.

If you use Web Hooks, then you must install each Web Hook event that you want on each of your text-enabled lines. If you have a single text-enabled line and you want to use all the Messaging API features, then you need to install all five Web Hook events on that line.

When you install a Web Hook event, you can install only a single event on one line. You cannot install an event on multiple lines with a single request. Also, you cannot install multiple events on a single line with a single request. Using the five-line example, if you want to install all five Web Hook events on all five-lines, then you would send a total of 25 install requests (five events x five lines = 25 requests).
4.5.1 Example: Web Hooks install (add) request

In the following example, the request is to add receive to the specified line.

```bash
$ curl -X POST https://api.zipwhip.com/webhook/add \
   -d session=[sessionKey] \
   -d type=message \
   -d event=receive \
   -d url=https://test.zipwhip.com/message/receive \
   -d method=POST
```

4.5.2 Table: Web Hooks installation parameters

The table below includes required and optional output parameters for installing Web Hooks on a text-enabled telephone number.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>Session is the response of user/login.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>type</td>
<td>For now, the available option is 'message'</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>event</td>
<td>One of the following events is required:</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>send</td>
<td>Transmits ZO message from text-enabled number.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>progress</td>
<td>Transmits internal message that identifies the status of the message.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>receive</td>
<td>Receives MO text message.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>read</td>
<td>Identifies whether message has been read (true) or not read (false).</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>delete</td>
<td>Removes the selected message from the message database.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>url</td>
<td>When events occur on the specified telephone number, this is the destination (web address) to which responses are sent.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>method</td>
<td>HTTP Method used for Web Hook (POST or PUT); Zipwhip uses POST as the default.</td>
<td>String</td>
<td>No</td>
</tr>
</tbody>
</table>
4.5.3 Example: Web Hooks installation response

The following is an example of the response to a successful installation of Web Hooks on a text-enabled telephone number. Note that in the example below, the method is POST when a message is received.

```json
{
    "success": true,
    "response": [
        {
            "webhookId": 687557714,
            "type": "message",
            "event": "receive",
            "url": "https://test.zipwhip.com/message/receive",
            "method": "POST"
        }
    ]
}
```

4.6 List Web Hooks

You can use the list request to view all of the Web Hooks that are currently installed for a given account.

4.6.1 Example: Web Hooks list request

The following example has the only required parameter, which is the session.

```
$ curl -G https://api.zipwhip.com/webhook/list \
    -d session=[sessionKey]
```

4.6.2 Table: Web Hooks list parameters

The only parameter required for a list request is a valid session key.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>Session is the response of user/login.</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>
4.6.3 Example: Web Hooks list response

The following is an example of the response to a list request. Note that the account has only one Web Hook event, which is receive. This means that the listed line is set up to use the method PUT to forward the message details the URL.

```json
{
    "success": true,
    "response": [
      {
        "webhookId": 687557714,
        "type": "message",
        "event": "receive",
        "url": "https://test.zipwhip.com/message/receive",
        "method": "PUT"
      }
    ]
}
```

4.7 Update Web Hooks

You can use the update request to change the existing Web Hook configuration on a given number. You can change the URL or change the Method used. You can only update Web Hooks individually. The webhookId identifies the specific Web Hook being updated.

4.7.1 Example: Web Hooks update request

The following is an example of a request to update the Web Hook events installed to a given text-enabled telephone number.

```
$ curl -X POST https://api.zipwhip.com/webhook/update \
   -d session=[sessionKey] \ 
   -d method=PUT \ 
   -d webhookID=687557714 
```
4.7.2 Table: Web Hooks update parameters

The table below includes required and optional parameters for updating Web Hooks on a text-enabled telephone number. You can update only one parameter in a given call.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>The response of user/login.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>webhookId</td>
<td>The Id of an installed Web Hook.</td>
<td>Integer</td>
<td>Yes</td>
</tr>
<tr>
<td>url</td>
<td>The destination when the event fires.</td>
<td>String</td>
<td>No</td>
</tr>
<tr>
<td>method</td>
<td>HTTP Method used for Web Hook (POST or PUT); Zipwhip uses POST as the default.</td>
<td>String</td>
<td>No</td>
</tr>
</tbody>
</table>

4.7.3 Example: Web Hooks update response

The following is an example of a successful Web Hooks update request. Note that the response is confirming the update to perform a PUT when a message-receive event occurs.

```json
{
  "success": true,
  "response": [
    {
      "webhookId": 687557714,
      "type": "message",
      "event": "receive",
      "url": "https://test.zipwhip.com/message/receive",
      "method": "PUT"
    }
  ]
}
```
4.8 Delete Web Hooks

You can send a delete request to remove an event from your Web Hooks configuration. You can delete one event at a time or delete multiple events in a single request.

4.8.1 Example: Web Hooks delete request

The following is an example of a request to delete the Web Hook events installed to a text-enabled telephone number.

```bash
$ curl -X POST https://api.zipwhip.com/webhook/delete \
    -d session=[sessionKey] \
    -d webhookId=687557714
```

4.8.2 Table: Web Hooks delete parameters

The table below includes required and optional parameters for deleting Web Hooks on a text-enabled telephone number.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>The response of user/login.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>webhookId</td>
<td>The Id of an installed Web Hook.</td>
<td>Integer</td>
<td>Yes</td>
</tr>
</tbody>
</table>

4.8.3 Example: Web Hooks delete response

The following is an example of a response to a successful Web Hooks delete request.

```json
{
    "success": true,
    "response": []
}
```

4.9 Single Host Versus Multiple Hosts

The primary difference between single-host and multiple-host implementations is that multiple-host implementations require different send, receive, and retry queues. Specifically, there are different retry queues for each host. If there are two hosts and one host fails, then Zipwhip routes traffic to the operating host. After rerouting the traffic, Zipwhip continues to attempt to send messages to the failed host, using an exponential back-off logarithm similar to that described in the Retry Logic section.

**Note:** If you have multiple hosts set up for an event, you are responsible for de-duplicating events.
5.0 Send a Scheduled Text Message

The second message-send scenario is setting up and sending a scheduled text message. Scheduled text messages can be sent to only one contact; you cannot send a scheduled message to multiple contacts. Scheduled text messages are different from text-only messages and text with image messages in two important ways: scheduled messages include the scheduleDate parameter, and use contacts as the parameter to identify the recipient.

To successfully send a scheduled text message, you must include the scheduleDate parameter. The message is sent only when the scheduledDate is reached, which can be set between a minimum of five (5) minutes in the future and a maximum of 365 days in the future. If you decide to make scheduled messages available to Operators, you must develop a mechanism for them to select the date on which they want the message sent.

To successfully send a scheduled text message, you must also use the contacts parameter to identify the recipient(s) of the scheduled message. If you do not use contacts, then your message is not sent and you receive an error message.

5.1 Using SMS to Send a Scheduled Text Message

This scenario is in a separate section because you must use a message request (which uses the SMS protocol) to send a scheduled text message. You can use either message or messaging to send a text-only message or text message with an image. However, to successfully send a scheduled text message, you must use message. If you use messaging to try and send a scheduled message, your message is not sent.

5.1.1 Example: Scheduled text message

The following is an example of a scheduled ZO text message to a single contact. Note the optional scheduledDate parameter:

```
$ curl https://api.zipwhip.com/message/send \
    -d session=3d0f1dde-aaff-4ce8-b61a-af212a860abc:123456789 \
    -d --data-urlencode contacts='8559479447' \
    -d --data-urlencode body='Body of a scheduled message!' \
    -d scheduledDate=1447707527000
```
5.1.2 Table: Scheduled message input parameters

The table below includes all of the required parameters that you can use when sending a scheduled ZO text message. The API will not support additional input parameters that you may develop.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>The GUID representation of a logged-in user.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>contacts</td>
<td>The String representation of a phone number.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>• US Domestic numbers are full, 10-digit numbers.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• International numbers must be in E.164 format.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>body</td>
<td>The message to be sent. The message body can be a maximum of 250 bytes.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>scheduledDate</td>
<td>Time/Date of desired delivery; must be in milliseconds and must be converted to Pacific Time.</td>
<td>String</td>
<td>No</td>
</tr>
</tbody>
</table>

5.1.3 Example: Scheduled message success response

The following is an example of the response to a scheduled ZO text message being sent successfully. This example includes both required parameters (highlighted) and optional parameters.

```
{
    "response": {
        "fingerprint": "4236521183",
        "root": "327559093363723008",
        "tokens": [
            {
                "contacts": 1989548603,
                "device": 309626613,
                "fingerprint": "42336654183",
                "message": "327545678963723008"
            }
        ]
    },
    "success": true
}
```
5.1.4 Table: Multiple-contact success output parameters

The table below includes required and optional output parameters included in the response to a scheduled ZO text message that is sent successfully.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>contacts</td>
<td>ContactId of the message recipient.</td>
<td>Integer</td>
<td>Yes</td>
</tr>
<tr>
<td>device</td>
<td>UserId of the message sender.</td>
<td>Integer</td>
<td>Yes</td>
</tr>
<tr>
<td>fingerprint</td>
<td>The fingerprint is the ConversationId, which is an ID assigned based on the recipients included with the outgoing text message.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>message</td>
<td>The ID of the message that was just sent.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>success</td>
<td>The success/failure of the web request</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

5.1.5 Example: Scheduled message failure response

The following is an example of the response to a ZO text message that was not sent successfully. This example includes all of the required parameters (highlighted) and optional parameters.

```
{
    "response": {
        "code": -703,
        "desc": "Bad arguments"
    },
    "success": false
}
```
### 5.1.6 Table: Scheduled message failure output parameters

The table below includes required and optional output parameters for a multiple-contact ZO text message that was not sent successfully.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>The Zipwhip error code.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>desc</td>
<td>Description of the error.</td>
<td>Integer</td>
<td>Yes</td>
</tr>
<tr>
<td>success</td>
<td>The success/failure of the web request. True=success; False = failure.</td>
<td>Boolean</td>
<td>No</td>
</tr>
</tbody>
</table>
6.0 Send an MMS (Image) Message

The third message-send scenario is sending a single-contact text message with an attached image. The process the Zipwhip API uses to send a message with an image attached is designed to minimize the message payload until the image is actually sent. To accomplish this, the API uses a multi-step process to send the message and the image:

1. When the Operator attaches the image, the API retrieves the image that the user/Operator wants to attach to the message. The API also notifies the internal host that the message does/does not (true/false) have an attached image.

2. One of the following occurs:

   • If the message does not have an attached image (false), then the message is processed as a text-only message. The API does not process the message to verify that an image is attached.
   • If the message does have an attached image (true), then the message is processed as a text message with an image.

3. When the user/Operator sends the message with the image, the internal host verifies that the message includes the attached image and that the image is correctly formatted. If there is no attached image or the format is incorrect, then the message is not sent, and the internal host sends an error message.

4. If the message has an attached image and image has the correct format, then the message is sent.

The maximum payload is 600kb. There is no transcoding functionality. The message body is limited to 700 bytes. Images larger than 700 bytes should be sent as an attachment. You can attach images with the following formats. Note that the final image format is text. You can use this to send large blocks of text:

   • image/bmp
   • image/gif
   • image/jpeg
   • image/png
   • text/plain

6.1 Using MMS to Send a Text Message with an Image

This scenario is in a separate section because you must use a messaging request (which uses the MMS protocol) to send a text message with an image. If you use message to send a message with an image, your message is not sent; however, no error message is sent. You must use messaging to successfully send messages with images.
6.2 Image Retrieve Process

There are two methods you can use for retrieving images: you can use the list-request method or the hosted-content method. The list-request method is described in this section; this method uses an attachment list request to retrieve the image(s). The hosted-content method is a proprietary method developed by Zipwhip. For more information about the hosted-content method, see Section 6.3 Retrieve Hosted Content.

When the Admin/Operator attaches an image to a message, the API sends an image retrieve request. This request has Attachment field, which is a Boolean value. If the value is true, then the API makes a messageAttachment/list web call to retrieve the appropriate images.

6.2.1 Example: MMS retrieve request

The following is an example of a web call for the list of images (attachments) that are being sent with the text message.

```
$ curl -G https://api.zipwhip.com/messageAttachment/list \
   -d session=[sessionKey] \
   -d messageId=[id of message]
```

6.2.2 Table: MMS retrieve parameters

The following table includes the required retrieve parameters.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>The GUID representation of a logged-in user.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>messageId</td>
<td>A unique String used to identify a given message object.</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>
6.2.3 Example: MMS retrieve response

In the following example response, the important values are the storageKey values (highlighted). These values are used in the examples that start in Section 6.4 Send a text message with an attached image.

```json
{
    "total":2,
    "response":[
        {
            "fileName":"IMG_1827.jpg",
            "dateCreated":"2015-03-26T13:15:17-07:00",
            "fileSizeBytes":40101,
            "mimeType":"image/jpeg",
            "storageKey":"be9396da-a6aa-442c-9406-asdfasdfsdf"
        },
        {
            "fileName":"123_1.smil",
            "dateCreated":"2015-03-26T13:15:17-07:00",
            "fileSizeBytes":300,
            "mimeType":"application/smil",
            "storageKey":"269023ec-67a0-4481-a820-asdfasdfsdf"
        }
    ]
}
```

6.3 Retrieve Hosted Content

Hosted Content is Zipwhip’s brand name for our proprietary MMS storage server. All items are stored with a key and permissions. You can use Hosted Content as a method for retrieving images to attach to text messages.

6.3.1 Example: MMS hosted content request

The following is an example of a Hosted Content image request:

```bash
$ curl -G https://api.zipwhip.com/hostedContent/get \
    -d session=[sessionKey] \ 
    -d storageKey=[storageKey]
```

6.3.2 Table: MMS retrieve hosted content parameters

The following table includes the required parameters for retrieving Hosted Content images:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>The GUID representation of a logged-in user.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>storageKey</td>
<td>A unique string used to identify a given media file.</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>
6.3.3 Example: MMS retrieve hosted content response

The following is an example of the response to a request for hosted content:

```bash
$ curl -G https://api.zipwhip.com/hostedContent/get \
    -d session=[sessionKey] \
    -d storageKey=[storageKey]
```

6.4 Send a Text Message with an Attached Image

When you send a text message with an attached image, you can attach one image, two images, or a large text attachment.

6.4.1 Example: Send a text message with one attached image

The following is an example of a text message with one attached image:

```bash
$ curl -X POST \
    -F "image=@Filename.png" \
  https://api.zipwhip.com/messaging/send"?"session=[sessionKey]"&"to=+12065551212"&"body=Hello
```

6.4.2 Example: Send a text message with two attached images

The following is an example of a text message with two attached images:

```bash
$ curl -X POST \
    -F "image=@Filename01.png" \
    -F "image=@Filename02.png" \
  https://api.zipwhip.com/messaging/send"?"session=[sessionKey]"&"to=+12065551212"&"body=Hello
```

6.4.3 Example: Send a text message with large text attachment

The following is an example of a text message with a large text attachment:

```bash
$ curl -X POST \
    -F "file=@textFile.txt" \
  https://api.zipwhip.com/messaging/send"?"session=[sessionKey]"&"to=+12065551212
```
6.4.4 Table: Image send parameters

The following table includes the required and optional parameters for sending a text message with an attached image:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>The response of user/login.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>to</td>
<td>The recipient’s phone number in E.164 format.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>body</td>
<td>Text body of the message (max 700 Bytes).</td>
<td>String</td>
<td>No</td>
</tr>
</tbody>
</table>

6.4.5 Example: Message with image success response

The following is an example of the response to a scheduled ZO text message with an attached image being sent successfully. This example includes required parameters.

```json
{
  "response": {
    "id": 664883126574915584,
    "status": "queued",
    "success": true
  }
}
```

6.4.6 Table: Message with image success response parameters

The following table includes the required and optional parameters for sending a text message with an attached image:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>The unique string assigned to the message.</td>
<td>Integer</td>
<td>Yes</td>
</tr>
<tr>
<td>status</td>
<td>The status of the message delivery.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>success</td>
<td>The success/failure of the web request. Success = true; Failure = false</td>
<td>Boolean</td>
<td>Yes</td>
</tr>
</tbody>
</table>
6.4.7 Example: Message with image failure response

The following is an example of the response to a ZO text message with an attached image that was not sent successfully. This example includes all of the required parameters. The Zipwhip API does not support additional parameters that you develop.

```json
{
    "success":false,
    "errorDesc": "Invalid recipients. Invalid phone numbers: +1206856. Phone numbers should be in e164 format."
}
```

6.4.8 Table: Message with image failure response parameters

The following table includes the required and optional parameters for sending a text message with an attached image:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td>The success/failure of the web request. True = success; False = failure.</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>errorDesc</td>
<td>Description of the error that caused failure.</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

6.4.9 Example: File too big failure response:

The following is an example of the response to a ZO text message with an attached image that failed because the attachment was too big.

```html
<html>
<head><title>413 Request Entity Too Large</title></head>
<body bgcolor="white">
<center><hr><h1>413 Request Entity Too Large</h1><hr></center>
<hr><center>nginx</center>
</body>
</html>
```
# 7.0 Status Code Summary

The status codes listed in this section apply to all message-send scenarios, including single-contact, multiple-contact, scheduled ZO messages, and image (MMS) messages.

## 7.1 Table: “Message Not Sent” Status Codes

If an error occurs when the ZO text message is sent, an error message appears. The current list of error codes is included in the table below.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>State</th>
<th>Retry</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1234</td>
<td>Recipient sent &quot;STOP&quot; message. Resolution = Recipient must send “Unblock” message.</td>
<td>Failure</td>
<td>No</td>
</tr>
<tr>
<td>-855</td>
<td>Internal Error</td>
<td>Failure</td>
<td>Yes</td>
</tr>
<tr>
<td>-801</td>
<td>Spambot Protection. Resolution = Contact <a href="mailto:api@zipwhip.com">api@zipwhip.com</a> to have spam flag removed.</td>
<td>Failure</td>
<td>No</td>
</tr>
<tr>
<td>-704</td>
<td>Internal Error</td>
<td>Failure</td>
<td>No</td>
</tr>
<tr>
<td>-703</td>
<td>Internal Error</td>
<td>Failure</td>
<td>No</td>
</tr>
<tr>
<td>-700</td>
<td>Internal Error</td>
<td>Failure</td>
<td>No</td>
</tr>
<tr>
<td>-394</td>
<td>Invalid Parameter Length. Resolution = May no longer be valid. May be replaced by -703/Bad Argument, which is caused by incorrect/invalid input parameter.</td>
<td>Failure</td>
<td>No</td>
</tr>
<tr>
<td>-124</td>
<td>Unable To Determine Carrier</td>
<td>Failure</td>
<td>No</td>
</tr>
<tr>
<td>-1</td>
<td>Unknown Error</td>
<td>Unknown</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### 7.2 Table: “Message Progress” Status Codes

When you use Web Hooks, the message-progress Status Codes allow you to track the progress of events as they occur. These specific codes are labeled as transient, so they are unlikely to be visible in any message/list API calls.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>State</th>
<th>Final State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Delivered – No Confirmation</td>
<td>Success</td>
<td>Yes</td>
</tr>
<tr>
<td>1</td>
<td>In Process</td>
<td>Transient</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>Queued</td>
<td>Transient</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>Queued</td>
<td>Transient</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>Delivered – Confirmed</td>
<td>Success</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>Error – Confirmed</td>
<td>Failure</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>Delivered – Confirmed by Delivery Receipt</td>
<td>Success</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>Error – Confirmed by Delivery Receipt</td>
<td>Failure</td>
<td>Yes</td>
</tr>
</tbody>
</table>