JavaScript SDK Documentation

Use the JavaScript SDK for the Machine Zone Real-Time Messaging (RTM) Service to create browser-based applications or server-based applications running within Node.js. The applications use the RTM Service to publish messages to channels and subscribe to channels.

For example, you can use the JavaScript SDK to create chat applications, create games that use the RTM Service to monitor game state, or create any other application utilizes the publish-subscribe messaging architecture implemented by the RTM Service. For an overview of the RTM Service system and components, see Real-Time Messaging (RTM) Service Overview.

Documentation for the JavaScript SDK includes the following topics:

- Getting Started with the JavaScript SDK. Basic steps to get started with the JavaScript SDK.
- JavaScript SDK Tutorial. Use the JavaScript SDK to create a lightweight browser-based chat application application.
- JavaScript SDK Reference. Get complete information about the classes and methods for the JavaScript SDK.

Getting Started with the JavaScript SDK

Complete the following basic steps to get started with the JavaScript SDK:

- Install the JavaScript SDK
- Create a connection to the RTM Service and create a channel object
- Subscribe to a channel
- Receive messages
- Publish to a channel

Note: The steps in this section assume you have already created an application object in the Developer Portal and have the appropriate application key to define the connection to the RTM Service. If you want to create a new application object, click **Apps** in the Developer Portal and click the **New App** button. See Creating an Application Object.

Installing the JavaScript SDK

The steps to install the JavaScript SDK depend on your development environment.

You can obtain the JavaScript SDK as a JavaScript file from Machine Zone, and then include the JavaScript file in your application file.

For example, insert the following code into an HTML file:

```
<!-- Include JavaScript SDK -->
<script src="path/to/file/sdk.min.js"></script>
```

Creating a Connection Object and Channel Object

To access the RTM Service with JavaScript, create a connection to the RTM Service with the RTM class create(appkey[, options]) method. Use the application key with the role of **public** that appears on the **Appkeys** tab of the **App Info** page for your application object in the Developer Portal.

Then, use the createChannel(name) method to create a new Channel object that represents the channel to which you want to subscribe or publish. If the channel does not already exist, the RTM Service creates it.

Use the following JavaScript code:

```
// create Connection object
var connection = MZ.RTM.create("46165ddB56C568aDFfd8860f08Baa03c"); // substitute your application key

// create Channel object
var channel = connection.createChannel("mzdemo.chat");
```

Subscribing to a Channel

To subscribe to messages for a specific channel, use the Channel class subscribe([next, isConfirmationRequired]) method.

To subscribe to a channel, you must have already created a Channel object with the createChannel(name) method. See Creating a Connection Object and Channel Object.

```
// subscribe to receive published messages
channel.subscribe();
```

Receiving Messages

To receive and process messages, create an event handler using the Channel class on(event, handler) method.

The following example executes the `showmessage` function on each message received on the subscribed channel:

```
// register data handler that executes on published messages
channel.on("data", function(data) {
    data.body.messages.forEach(showMessage);
});
```

Publishing to a Channel

To publish messages to a channel, use the Channel class publish(message[, isConfirmationRequired]) method.

To publish messages, you must have already created a Channel object with the createChannel(name) method. See Creating a Connection Object and Channel Object.

```
channel.publish({message: "Message Text"});
```

JavaScript SDK Tutorial

This tutorial shows the following basic steps to use the Developer Portal and the JavaScript SDK to create a browser-based chat application.

This JavaScript application connects to the Real-Time Messaging (RTM) Service to send and receive messages. After you run the application, you can view statistics for the application in the Developer Portal.

Note: Because this is a browser-based application, you do not need to upload the application to a web server to run it. However, you will need to share the file with other users to chat in real-time using the application.

Tutorial Steps

Task	Location
Step 1. Create JavaScript Application Source	Text Editor
Step 2. Run Chat Application	Browser
Step 3. View Statistics	Developer Portal

Before You Begin

To complete this tutorial, you will need:

- Developer Portal account with an application object created
- Text editor
- Chrome browser

Step 1. Create the JavaScript Application Source

Machine Zone provides RTM client SDKs to create client applications that use the RTM Service to subscribe and publish to channels.

In this step, you use the methods in the JavaScript SDK to create a chat application.

Full JavaScript Chat Application Source

Include the following code sample in a text editor or IDE of your choice and save the file with a **.htm** or **.html** extension.

Note: Replace `<Application Key>` in the following line with the application key with the **Public** role that appears on the **Appkeys** subtab for your application:

```
var connection = MZ.RTM.create("<Application Key>");
```

```
<!DOCTYPE html>
<html>
    <head>
        <title>RTM/JavaScript Chat Application</title>

        <!-- CSS styles -->
        <style>
            input.message {
                border: 1px solid #ccc;
                border-radius: 4px 0px 0px 4px;
                padding: 6px 12px;
                font-size: 14px;
                margin-right: 0px;
                outline: none;
            }
            span.message {
                color: white;
                background: linear-gradient(#1ad6fd , #1d62f0);
                padding: 5px 10px;
                border-radius: 15px;
            }
            div.message {
                margin: 15px 0px;
            }
            .send {
                font-size: 14px;
                padding: 6px 12px;
                margin-left: -5px;
                background-color: #fff;
                border: 1px solid #ccc;
                border-radius: 0px 4px 4px 0px;
                cursor: pointer;
            }
        </style>
    </head>
    <body>

        <!-- RTM JavaScript library -->
        <script src="http://cdn.platform.machinezone.com/rtm-js-sdk/v0.5.0/sdk.min.js"></script>

        <h3>Chat demo app powered by RTM</h3>
        <div id="stream" class="stream"></div>
        <input type="text" id="message" class="message" placeholder="Type your message..."/>
        <button id="send" class="send">Send</button>

        <!-- JavaScript to communicate with RTM Service and run chat application -->
        <script>

            // create PubSub instance
            var connection = MZ.RTM.create("<Application Key>");

            // create Channel object
            var channel = connection.createChannel("mzdemo.chat");

            // register data handler that executes on published messages
            channel.on("data", function(data) {
                data.body.messages.forEach(showMessage);
            });

            // subscribe to receive published messages
            channel.subscribe();


            <!-- Chat application logic -->
            var message = document.getElementById("message");
            var sendButton = document.getElementById("send");
            sendButton.addEventListener("click", function() {
            if (!message.value) {
                message.focus();
                return;
            }
                channel.publish({"message": message.value});
                message.value = "";
                message.focus();
            });
            message.addEventListener("keydown", function(event) {
                if (event.keyCode === 13) {  // Enter
                    sendButton.click();
                }
            });
            var showMessage = function(data) {
            var text = data.message;
            var stream = document.getElementById("stream");
            var text = '<span class="message">' + text + '</span>';
            var item = document.createElement("div");
            item.className = "message";
            item.innerHTML = text;
                stream.appendChild(item);
            };

    </script>
    </body>
</html>
```

Back to JavaScript SDK Tutorial

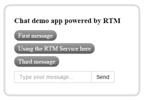
Step 2. Run Chat Application

In a Chrome browser, open the HTML file you created in Step 2. Create the JavaScript Application Source. The chat application connects to the RTM Service to publish messages and receive published messages:

You can use, for example, one of the following methods to use the chat application with other users:

- Upload the source file to a Google Docs drive and share the URL.
- Upload the source file to a web server and share the URL.

The following screenshot shows the results of a chat session using the application.
[Click image]



Chat demo app powered by RTM

First message

Using the RTM Service here

Third message

Step 3. View Statistics

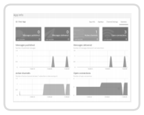
As you use the chat application, the Developer Portal displays a stream of real-time statistics about the messages and channels in the application.

In this step, you view the statistics collected by the Developer Portal as you use the chat application.

To view the real-time statistics for your application, click **Dashboard** in the navigation panel.

The following screenshot shows sample statistics for the application in the Developer Portal.

[Click image]



Back to JavaScript SDK Tutorial

JavaScript SDK Reference

The JavaScript SDK includes two classes: RTM and Channel.

Class Name	Description
RTM	Manages the WebSocket connection to the RTM Service. Use the methods in this class to connect to the RTM Service, create channels, and open and close the RTM Service connection.
Channel	Represents a channel and handles channel operations, including publishing and subscribing, and handling channel events.

RTM Class

The RTM class manages the WebSocket connection to the RTM Service. Use the methods in this class to connect to the RTM Service, create channels, and open and close the connection to the RTM Service.

Method	Description
create(appkey[, options])	Returns an instance of a WebSocket connection to the RTM Service as an object.
createChannel(name)	Creates a channel and returns an instance of the Channel class as an object. Use this method once for each channel to which you want to subscribe or publish.
open()	Reopens a WebSocket connection to the RTM Service for an existing connection object created with the create(appkey[, options]) method and returns the connection object.
close()	Closes a connection to the RTM Service. Returns a Promise object that resolves after the connection has been closed or rejects if the connection is already closed.
on(event, fn)	Specifies the handler function to execute when a specific action on the connection occurs and returns an instance of the connection object.
off(id)	Removes a previously attached handler from the connection object and returns an instance of the connection object.

create(appkey[, options])

Description	Returns an instance of a WebSocket connection to the RTM Service as an object.

This method only creates the object. The actual WebSocket connection to the RTM Service occurs during the first publish or subscribe operation. In addition, if you create multiple connection objects with this method, the SDK uses a single WebSocket connection to the RTM Service to manage all connection objects.

You can use the `options` parameter to specify additional properties for the connection object. The `maxRetries` and `maxRetriesTimeout` properties apply to publish and subscribe operations that use that connection object. Otherwise, the default values are used for each connection.

Returns `Object`

Parent Class RTM

Parameters

- `appkey` {string} [required] - Application key for the connection to the RTM Service. This key is located on the **Appkeys** tab of the **App Info** page for your application object in the Developer Portal, with a role of **public**.
- `options` {object} [optional] - Object used to specify additional properties for the connection.
- `options.url` {string} [optional] - Defines the URL of the RTM Service endpoint for the connection. Default value for this property is `wss://api.platform.machinezone.com`.
- `options.maxRetries` {int} [optional] - Number of times the method attempts to reconnect to the RTM Service before an operation fails. Default is 10.
- `options.maxRetriesTimeout` {int} [optional] - Time period, in seconds, between reconnection attempts. The timeout period between each successive connection attempt increases, but will never exceed this value.

Throws

- `TypeError`: Thrown if required parameters are missing or invalid.

Syntax

```
// create Connection object
var connection = MZ.RTM.create("46165ddB56C568aDFfd8860f08Baa03c"); // substitute your application key
```

createChannel(name)

Description Creates a channel and returns an instance of the Channel class as an object.

Use this method once for each channel to which you want to subscribe or publish.

Returns `Object`

Parent Class RTM

Parameters

- `name` {string} [required] - Name of the channel to which you want to subscribe or publish.

Throws

- `TypeError`: Thrown if the name parameter is missing or invalid.

Syntax

```
// create Connection object
var connection = MZ.RTM.create("46165ddB56C568aDFfd8860f08Baa03c"); // substitute your application key
var channel = connection.createChannel("channel name");
```

open()

Description Reopens a WebSocket connection to the RTM Service for an existing connection object created with the create(appkey[, options]) method. Returns a Promise object that resolves after the connection has been opened.

Use this method only if you closed a connection using the close() method. Otherwise, the WebSocket connection with the RTM Service is established when the RTM Service performs the first publish or subscribe operation for the channel.

Returns `void`

Parent Class RTM

Syntax

```
// create Connection object
var connection = MZ.RTM.create("46165ddB56C568aDFfd8860f08Baa03c"); // substitute your application key
...
connection.close(); // connection explicitly closed
...
connection.open(); // reopen the connection
```

close()

Description Closes a connection to the RTM Service. Returns a Promise object that resolves after the connection has been closed or rejects if the connection is already closed.

This method deletes all channel objects created for the connection. Use this method if you want to explicitly close all interaction with the RTM Service created by the connection object.

Returns `Object`

Parent Class RTM

Syntax

```
// create Connection object
var connection = MZ.RTM.create("46165ddB56C568aDFfd8860f08Baa03c"); // substitute your application key
// additional logic ...
connection.close(); // connection explicitly closed
```

on(event, fn)

Description Specifies the handler function to execute when a specific action on the connection occurs and returns an instance of the connection object. You can create functions for the following event types for the `event` parameter:

- `open`: The connection is opened.
- `close`: The connection was closed.

- `beforeRetry`: The application handled a connection error and will reconnect to the RTM Service.
- `retry`: The application attempts to reconnect to the RTM Service.
- `error`: An error occurred with the connection.

> **Note:** You must use a named function for the `fn` parameter if you want to use the off([name, fn]) function to remove it at a later point.

Returns `Object`

Parent Class RTM

Parameters

- `event` {string} [required] - Type of event for the handler function.
- `function` {function} [required] - JavaScript function to define application functionality that occurs on the `event` parameter.

Throws

- `TypeError`: Thrown if the `name` or `fn` parameter is missing or invalid.

Syntax

```
var connection = MZ.RTM.create("46165ddB56C568aDFfd8860f08Baa03c"); // substitute your application key
// register an "error" handler...
connection.on("error", function() {
    // handler code here
});
// register a function to be executed once connection is opened...
connection.on("open", function() {
    // handler code here
});
```

off([event, fn])

Description Removes a previously attached handler from the connection object and returns the connection object instance.

You can remove a specific event with the `event` parameter and a specific function with the `fn` parameter. If you only use the `event` parameter, the SDK removes event handlers for that event type.

If you do not specify these parameters, the SDK removes all event handlers from the connection object.

Returns `object`

Parent Class RTM

Parameters

- `event` {string} [optional] - Name of the event type to remove. See on(event, fn).
- `function` {string} [optional] - Function name to remove.

Throws

- `TypeError`: Thrown if the `id` parameter is missing or invalid.

Syntax

```
var connection = MZ.RTM.create("46165ddB56C568aDFfd8860f08Baa03c"); // substitute your application key
var errorHandler = function() {
    // your handler code here
};
// register an "error" handler...
channel.on("error", errorHandler);
// detach handler when you no longer need it...
channel.off("error", errorHandler);
// detach all "error" handlers...
channel.off("error");
// detach all handlers
channel.off();
```

Channel Class

The Channel class represents a channel and handles channel operations, including publishing and subscribing, and handling channel events. This class does not require a constructor, use the createChannel(name) method instead.

Method	Description
publish(message[, isConfirmationRequired])	Publishes a message into a channel. If you use the `isConfirmationRequired` parameter, returns a Promise object that resolves after the publish operation succeeds or fails.
subscribe([next, isConfirmationRequired])	Subscribes to a channel and receives all messages published to that channel. If you use the `isConfirmationRequired` parameter, returns a Promise object that resolves after the subscribe operation succeeds or fails.
unsubscribe([isConfirmationRequired])	Unsubscribes from all messages for a channel. If you use the `isConfirmationRequired` parameter, returns a Promise object that resolves after the unsubscribe operation succeeds or fails.
close()	Closes the WebSocket connection for the associated Channel object. This method unsubscribes from the channel, removes all event handlers, and closes the WebSocket connection. If you use the `waitUnsubcribeConfirmation` parameter, this method returns a Promise object that resolves after the all the actions of the method complete, or fails if the WebSocket connection is already closed.
on(event, fn)	Specifies the handler function to execute when a specific event on the Channel object occurs and returns the Channel object instance.
off(id)	Removes a previously attached handler function from the Channel object and returns the Channel object instance.

publish(message[, isConfirmationRequired])

Description Publishes a message into a channel. If you use the isConfirmationRequired parameter, returns a Promise object that resolves after the publish operation succeeds or fails.

The `message` parameter can be any JSON-supported value. For more information, see www.json.org (http://www.json.org).

The `isConfirmationRequired` parameter directs the RTM Service to confirm the status of the publish operation. If the RTM Service publishes the message within ten seconds of sending the request, the operation suceeds. Otherwise, the operation fails. Use this parameter if subsequent operations depend on the result of the publish operation.

Note: The first publish or subscribe operation to a channel establishes the WebSocket connection to the RTM Service.

Returns `Object`

Parent Class Channel

`Parameters`

- `message` {string} [required] - JSON value to publish.
- `isConfirmationRequired` {boolean} [optional] - Directs the RTM Service to confirm the status of the publish operation. Default value is `false`.

Throws

- `TypeError`: Thrown if the `message` parameter is invalid.

Syntax

```
var connection = MZ.RTM.create("46165ddB56C568aDFfd8860f08Baa03c"); // substitute your application key
var channel = connection.createChannel("your-channel");

// publish with no confirmation (default)...
channel.publish({key: value});

// publish with confirmation...
channel.publish({key: value}, true)
    .then(function() {
        // publish operation succeeds
    }, function() {
        // operation failed or no confirmation seen within 10 seconds
    });
```

subscribe([next, isConfirmationRequired])

Description Subscribes to a channel and receives all messages published to that channel. If you use the `isConfirmationRequired` parameter, returns a Promise object that resolves after the subscribe operation succeeds or fails.

Use the `next` parameter to specify the string value of the last known channel stream position prior to the subscribe operation. For more information, see Using the Next Parameter.

The `isConfirmationRequired` parameter directs the RTM Service to confirm the status of the subscribe operation. If the RTM Service subscribes to the channel within ten seconds of sending the request, the operation suceeds. Otherwise, the operation fails. Use this parameter if subsequent operations depend on the result of the the subscribe operation.

Note: The first publish or subscribe operation to a channel establishes the WebSocket connection to the RTM Service.

Returns `Object`

Parent Class Channel

Parameters

- `next` {string} [optional] **-** String value of the last known channel stream position before the subscribe operation. If necessary, you can skip this parameter and use `isConfirmationRequired` only.
- `isConfirmationRequired` {boolean} [optional] **-** Directs the RTM Service to confirm the status of the subscribe operation. Default value is `false` .

Syntax

```
var connection = MZ.RTM.create("46165ddB56C568aDFfd8860f08Baa03c"); // substitute your application key
var channel = connection.createChannel("your-channel");

// subscribe with no confirmation (default)...
channel.subscribe();
// subscribe with next and confirmation...
channel.subscribe(7862334, true)
    .then(function() {
        // everything is ok
    }, function() {
        // operation failed or no confirmation seen within 10 seconds
    });

// subscribe with confirmation only...
channel.subscribe(true)
    .then(function() {
        // everything is ok
    }, function() {
        // operation failed or no confirmation seen within 10 seconds
    });
```

unsubscribe([isConfirmationRequired])

Description Unsubscribes from all messages for a channel. If you use the `isConfirmationRequired` parameter, returns a Promise object that resolves after the unsubscribe operation succeeds or fails.

The `isConfirmationRequired` parameter directs the RTM Service to confirm the status of the unsubscribe operation. If the RTM Service unsubscribes from the channel within ten seconds of sending the request, the operation suceeds. Otherwise, the operation fails. Use this parameter if subsequent operations depend on the result of the unsubscribe operation.

Note: If you want, you can subscribe to this channel again after unsubscribing. If you want to unsubscribe and close the WebSocket connection, use close([waitUnsubscribeConfirmation]) instead.

Returns `Object`

Parent Class Channel

Parameters

- `isConfirmationRequired` {boolean} [optional] - Directs the RTM Service to confirm the status of the unsubscribe operation. Default value is `false`.

Syntax

```
var connection = MZ.RTM.create("46165ddB56C568aDFfd8860f08Baa03c"); // substitute your application key
var channel = connection.createChannel("your-channel");

// unsubscribe with no confirmation (default)...
channel.unsubscribe();

// unsubscribe with confirmation...
channel.unsubscribe(true)
    .then(function() {
        // everything is ok
    }, function() {
        // operation failed or no confirmation seen within 10 seconds
    });
```

close([waitUnsubscribeConfirmation])

Description Closes the WebSocket connection for the associated Channel object. This method unsubscribes from the channel, removes all event handlers, and closes the WebSocket connection. If you use the `waitUnsubcribeConfirmation` parameter, this method returns a Promise object that resolves after the all the actions of the method complete, or fails if the WebSocket connection is already closed.

Use the `waitUnsubcribeConfirmation` parameter if subsequent operations depend on the result of the close operation.

Note: To subscribe to the channel after calling this method, you must use the createChannel(name) method again. If you want to only unsubscribe from the channel, use the unsubscribe([isConfirmationRequired]) method instead.

Returns `Object`

Parent Class Channel

Parameters

- `waitUnsubscribeConfirmation` {boolean} [optional] - Directs the RTM Service to confirm the status of the close operation. Default value is `false`.

Syntax

```
var connection = MZ.RTM.create("46165ddB56C568aDFfd8860f08Baa03c"); // substitute your application key
var channel = connection.createChannel("your-channel");

// closing a channel with confirmation of unsubscribe operation (default)...
channel.close();

// closing a channel with no confirmation of unsubscribe operation...
channel.close(false);
```

on(event, fn)

Description Specifies the handler function to execute when a specific event on the Channel object occurs and returns the Channel object instance.

You can create functions for the following event types for the `event` parameter:

- `data`: A new message is published to the channel.
- `close`: The channel is closed via the close([waitUnsubscribeConfirmation]) method or the connection is closed via the close() method on the RTM object.
- `error`: An error occurred during communication on the channel with the RTM Service.
- `beforeRetry`: The RTM Service handles an error and before the RTM Service attempts to reconnect.
- `retry`: The RTM Service handled an error and attempts to reconnect to the RTM Service.

Note: You must use a named function for the `fn` parameter if you want to use the off([name, fn]) function to remove it at a later point.

Returns `Object`

Parent Class Channel

Parameters

- `event` {string} [required] - Type of event for the handler function.
- `fn` {object} [required] - JavaScript function to define application functionality that occurs on the `event` parameter.

Throws

- `TypeError`. Thrown if the `event` or `fn` parameter is missing or invalid.

Syntax

```
var connection = MZ.RTM.create("46165ddB56C568aDFfd8860f08Baa03c"); // substitute your application key
var channel = connection.createChannel("your-channel");

// register an "error" handler...
channel.on("error", function() {
    // your handler code here
});

// register a function to be executed if there is data published to the channel
channel.on("data", function() {
    // your handler code here
});
```

off([event, fn])

Description Removes a previously attached handler function from the Channel object and returns the Channel object instance.

You can remove a specific event with the `event` parameter and a specific function with the `fn` parameter. If you only use the `event` parameter, the SDK removes the event handlers for that event type.

If you do not specify any parameter, the SDK removes all event handlers from the Channel object.

Returns `Object`

Parent Class Channel

Parameters

- `event` {string} [optional] - Name of the handler event type to remove. See on(event, fn).
- `fn` {string} [optional] - Function name to remove.

Throws

- `TypeError`. Thrown if the `event` parameter is invalid.

Syntax

```
var connection = MZ.RTM.create("46165ddB56C568aDFfd8860f08Baa03c"); // substitute your application key
var channel = connection.createChannel("your-channel");

var errorHandler = function() {
    // your handler code here
};

// register an "error" handler...
channel.on("error", errorHandler);

// detach handler when you no longer need it...
channel.off("error", errorHandler);

// detach all "error" handlers...
channel.off("error");

// detach all handlers...
channel.off();
```