Notes

Firebase Authentication

See the Firebase Authentication docs for web.

onAuthStateChanged

```
firebase.auth().onAuthStateChanged(currentUser => {
  if (currentUser) {
    // User is signed in.
  } else {
    // No user is signed in.
  }
});
```

Register Email/Password

```
firebase.auth().createUserWithEmailAndPassword(email,
password).catch(function(error) {
  // Handle Errors here.
  var errorCode = error.code;
  var errorMessage = error.message;
  // ...
});
```

Sign In Email/Password

```
firebase.auth().signInWithEmailAndPassword(email, password).catch(function(error)
{
  // Handle Errors here.
  var errorCode = error.code;
  var errorMessage = error.message;
  // ...
});
```

Create Provider

Google

```
var provider = new firebase.auth.GoogleAuthProvider();
```

Facebook

```
var provider = new firebase.auth.FacebookAuthProvider();
```

Twitter

```
var provider = new firebase.auth.TwitterAuthProvider();
```

GitHub

```
var provider = new firebase.auth.GithubAuthProvider();
```

OAuth sign in with a provider

Popup

```
firebase.auth().signInWithPopup(provider);
```

Redirect

```
firebase.auth().signInWithRedirect(provider);
```

Phone Auth

First attach a recaptcha using an element ID...

```
window.recaptchaVerifier = new firebase.auth.RecaptchaVerifier('sign-in-button', {
  'size': 'invisible',
  'callback': function(response) {
    // reCAPTCHA solved, allow signInWithPhoneNumber.
    onSignInSubmit();
  }
});
```

... then capture a phone number from user input and send the sms...

```
var phoneNumber = getPhoneNumberFromUserInput();
var appVerifier = window.recaptchaVerifier;
firebase.auth().signInWithPhoneNumber(phoneNumber, appVerifier)
  .then(function (confirmationResult) {
    // SMS sent. Prompt user to type the code from the message, then sign the
    // user in with confirmationResult.confirm(code).
    window.confirmationResult = confirmationResult;
  }).catch(function (error) {
    // Error; SMS not sent
    // ...
  });
```

...and finally authenticate with the code from user input.

```
var code = getCodeFromUserInput();
confirmationResult.confirm(code).then(function (result) {
  // User signed in successfully.
  var user = result.user;
  // ...
}).catch(function (error) {
  // User couldn't sign in (bad verification code?)
  // ...
});
```

Notes

Cloud Firestore

See the Cloud Firestore docs for web.

Set a document

```
var data = {
  name: 'Los Angeles',
  state: 'CA',
  country: 'USA',
};

// Add a new document in collection "cities" with ID 'LA'
var setDoc = db
  .collection('cities')
  .doc('LA')
  .set(data);
```

Data types

```
var data = {
  stringExample: 'Hello, World!',
  booleanExample: true,
  numberExample: 3.14159265,
  dateExample: new Date('December 10, 1815'),
  arrayExample: [5, true, 'hello'],
  nullExample: null,
  objectExample: {
    a: 5,
    b: true,
  },
};

var setDoc = db
  .collection('data')
  .doc('one')
  .set(data);
```

Add document with auto-generated ID

> In a single-step with **asynchronous** access to the new ref

```
// Add a new document with a generated id.
var addDoc = db
  .collection('cities')
  .add({
    name: 'Tokyo',
    country: 'Japan',
  })
  .then(ref => {
    console.log('Added document with ID: ', ref.id);
  });
```

> In two steps with **synchronous** access to the new ref

```
// Add a new document with a generated id.
var newCityRef = db.collection('cities').doc();

console.log('newCityRef id:', newCityRef.id);

var setDoc = newCityRef
  .set({
    name: 'Tokyo',
    country: 'Japan',
  })
  .then(ref => {
    //...
  });
```

Update document

Note the optional `merge: true` option

```
var cityRef = db.collection('cities').doc('DC');

// Set the 'capital' field of the city
var updateSingle = cityRef.update({ capital: true }, { merge: true });
```

Transactions

```
// Initialize document
var cityRef = db.collection('cities').doc('SF');
var setCity = cityRef.set({
  name: 'San Francisco',
  state: 'CA',
  country: 'USA',
  capital: false,
  population: 860000,
});

var transaction = db
  .runTransaction(t => {
    return t.get(cityRef).then(doc => {
      // Add one person to the city population
      var newPopulation = doc.data().population + 1;
      t.update(cityRef, { population: newPopulation });
    });
  })
  .then(result => {
    console.log('Transaction success!');
  })
  .catch(err => {
    console.log('Transaction failure:', err);
  });
```

Batched writes

```
// Get a new write batch
var batch = db.batch();

// Set the value of 'NYC'
var nycRef = db.collection('cities').doc('NYC');
batch.set(nycRef, { name: 'New York City' });

// Update the population of 'SF'
var sfRef = db.collection('cities').doc('SF');
batch.update(sfRef, { population: 1000000 });

// Delete the city 'LA'
var laRef = db.collection('cities').doc('LA');
batch.delete(laRef);

// Commit the batch
return batch.commit().then(function() {
  // ...
});
```

Bulk delete

> Max batch size is 500 records

```
function deleteCollection(db, collectionPath, batchSize) {
  var collectionRef = db.collection(collectionPath);
  var query = collectionRef.orderBy('__name__').limit(batchSize);

  return new Promise((resolve, reject) => {
    deleteQueryBatch(db, query, batchSize, resolve, reject);
  });
}

function deleteQueryBatch(db, query, batchSize, resolve, reject) {
  query
    .get()
    .then(snapshot => {
      // When there are no documents left, we are done
      if (snapshot.size == 0) {
        return 0;
      }

      // Delete documents in a batch
      var batch = db.batch();
      snapshot.docs.forEach(doc => {
        batch.delete(doc.ref);
      });

      return batch.commit().then(() => {
        return snapshot.size;
      });
    })
    .then(numDeleted => {
      if (numDeleted === 0) {
        resolve();
        return;
      }

      // Recurse on the next process tick, to avoid
      // exploding the stack.
      process.nextTick(() => {
        deleteQueryBatch(db, query, batchSize, resolve, reject);
      });
    })
    .catch(reject);
}
```

Get a document

```
var cityRef = db.collection('cities').doc('SF');
var getDoc = cityRef
  .get()
  .then(doc => {
    if (!doc.exists) {
      console.log('No such document!');
    } else {
      console.log('Document data:', doc.data());
    }
  })
  .catch(err => {
    console.log('Error getting document', err);
  });
```

Get an entire collection

```
var citiesRef = db.collection('cities');
var allCities = citiesRef
  .get()
  .then(snapshot => {
    snapshot.forEach(doc => {
      console.log(doc.id, '=>', doc.data());
    });
  })
  .catch(err => {
    console.log('Error getting documents', err);
  });
```

Get with a where clause

```
var citiesRef = db.collection('cities');
var query = citiesRef
  .where('capital', '==', true)
  .get()
  .then(snapshot => {
    snapshot.forEach(doc => {
      console.log(doc.id, '=>', doc.data());
    });
  })
  .catch(err => {
    console.log('Error getting documents', err);
  });
```

List subcollections

```
var sfRef = db.collection('cities').doc('SF');
sfRef.getCollections().then(collections => {
  collections.forEach(collection => {
    console.log('Found subcollection with id:', collection.id);
  });
});
```

Listen for document changes

```
var doc = db.collection('cities').doc('SF');

var observer = doc.onSnapshot(
  docSnapshot => {
    console.log(`Received doc snapshot: ${docSnapshot}`);
    // ...
  },
  err => {
    console.log(`Encountered error: ${err}`);
  }
);
```

Listen for collection changes

```
var query = db.collection('cities').where('state', '==', 'CA');

var observer = query.onSnapshot(
  querySnapshot => {
    console.log(`Received query snapshot of size ${querySnapshot.size}`);
    // ...
  },
  err => {
    console.log(`Encountered error: ${err}`);
  }
);
```

Stop listening

```
var unsub = db.collection('cities').onSnapshot(() => {});

// ...

// Stop listening for changes
unsub();
```

Compound queries

> Valid queries

```
citiesRef.where('state', '==', 'CO').where('name', '==', 'Denver');

citiesRef.where('state', '==', 'CA').where('population', '<', 1000000);

citiesRef.where('state', '>=', 'CA').where('state', '<=', 'IN');

citiesRef.where('state', '==', 'CA').where('population', '>', 1000000);
```

> !!! INVALID QUERY AHEAD !!!

```
// Invalid query. Will throw an error.
citiesRef.where('state', '>=', 'CA').where('population', '>', 1000000);
```

Order and limit

> Valid order/limit combinations

```
var firstThree = citiesRef.orderBy('name').limit(3);

var lastThree = citiesRef.orderBy('name', 'desc').limit(3);

var byStateByPop = citiesRef.orderBy('state').orderBy('population', 'desc');

var biggest = citiesRef
  .where('population', '>', 2500000)
  .orderBy('population')
  .limit(2);

var allBigCities = citiesRef.where('population', '>',
2500000).orderBy('population');
```

> !!! INVALID QUERY AHEAD !!!

```
// Invalid query. Will throw an error.
citiesRef.where('population', '>', 2500000).orderBy('country');
```

Pagination: single-cursor

> Valid pagination

```
var startAt = db
  .collection('cities')
  .orderBy('population')
  .startAt(1000000);

var startAfter = db
  .collection('cities')
  .orderBy('population')
  .startAfter(1000000);

var endAt = db
  .collection('cities')
  .orderBy('population')
  .endAt(1000000);

var endBefore = db
  .collection('cities')
  .orderBy('population')
  .endBefore(1000000);
```

Pagination: multiple-cursors

```
// Will return all Springfields
var startAtName = db
  .collection('cities')
  .orderBy('name')
  .orderBy('state')
  .startAt('Springfield');

// Will return "Springfield, Missouri" and "Springfield, Wisconsin"
var startAtNameAndState = db
  .collection('cities')
  .orderBy('name')
  .orderBy('state')
  .startAt('Springfield', 'Missouri');
```

Notes

Realtime Database

See the Realtime Database docs for web.

Set a ref

```
function writeUserData(userId, name, email, imageUrl) {
  firebase
    .database()
    .ref('users/' + userId)
    .set({
      username: name,
      email: email,
      profile_picture: imageUrl,
    });
}
```

Value events

Value events fire with the entire data payload for any and all changes

> Listen to ongoing events

```
var starCountRef = firebase.database().ref('posts/' + postId + '/starCount');
starCountRef.on('value', function(snapshot) {
  updateStarCount(postElement, snapshot.val());
});
```

> Listen to a single event and stop listening

```
var userId = firebase.auth().currentUser.uid;
return firebase
  .database()
  .ref('/users/' + userId)
  .once('value')
  .then(function(snapshot) {
    var username = (snapshot.val() && snapshot.val().username) || 'Anonymous';
    // ...
  });
```

Multi-path updates

```
function writeNewPost(uid, username, picture, title, body) {
  // A post entry.
  var postData = {
    author: username,
    uid: uid,
    body: body,
    title: title,
    starCount: 0,
    authorPic: picture,
  };

  // Get a key for a new Post.
  var newPostKey = firebase
    .database()
    .ref()
    .child('posts')
    .push().key;

  // Write the new post's data simultaneously in the posts list and the user's
post list.
  var updates = {};
  updates['/posts/' + newPostKey] = postData;
  updates['/user-posts/' + uid + '/' + newPostKey] = postData;

  return firebase
    .database()
    .ref()
    .update(updates);
}
```

Delete data

```
function deleteUser(userId) {
  return firebase
    .database()
    .ref('/users/' + userId)
    .remove();
}
```

Detach listener

```
var starCountRef = firebase.database().ref('posts/' + postId + '/starCount');
var listener = starCountRef.on('value', function(snapshot) {
  updateStarCount(postElement, snapshot.val());
});

function detachListener() {
  starCountRef.off('value', listener);
}
```

Transactions

```
function toggleStar(postRef, uid) {
  postRef.transaction(function(post) {
    if (post) {
      if (post.stars && post.stars[uid]) {
        post.starCount--;
        post.stars[uid] = null;
      } else {
        post.starCount++;
        if (!post.stars) {
          post.stars = {};
        }
        post.stars[uid] = true;
      }
    }
    return post;
  });
}
```

Child events

- **child_added**: fires once for every existing result and then again for every new result; does not fire for changes or removals, only new records
- **child_changed**: fires when the underlying object or value is changed in any way
- **child_removed**: fires when the entire record is removed

```
var commentsRef = firebase.database().ref('post-comments/' + postId);
commentsRef.on('child_added', function(data) {
  addCommentElement(postElement, data.key, data.val().text, data.val().author);
});

commentsRef.on('child_changed', function(data) {
  setCommentValues(postElement, data.key, data.val().text, data.val().author);
});

commentsRef.on('child_removed', function(data) {
  deleteComment(postElement, data.key);
});
```

Sort data

- **orderByChild('childName')**: Orders by a child attribute
- **orderByKey()**: Orders by record keys
- **orderByValue()**: Orders by record values; only relevant when values are strings or numbers and not nested objects

```
var topUserPostsRef = firebase
  .database()
  .ref('user-posts/' + myUserId)
  .orderByChild('starCount');

var mostViewedPosts = firebase
  .database()
  .ref('posts')
  .orderByChild('metrics/views');
```

Filter data

> Assumes that data is ordered by key unless otherwise specified

- **limitToFirst(count)**: Sets the maximum number of items to return from the beginning of the ordered list of results.
- **limitToLast(count)**: Sets the maximum number of items to return from the end of the ordered list of results.
- **startAt(value)**: Return items greater than or equal to the specified key or value, depending on the order-by method chosen.
- **endAt(value)**: Return items less than or equal to the specified key or value, depending on the order-by method chosen.
- **equalTo(value)**: Return items equal to the specified key or value, depending on the order-by method chosen.

```
var first100Days = firebase
  .database()
  .ref('days/2018')
  .orderByChild('dayOfYear')
  .limitToFirst(100);

var first10DaysOfFebruary = firebase
  .database()
  .ref('days/2018')
  .orderByChild('dayOfYear')
  .limitToFirst(10)
  .startAt(32);

var last10DaysOfJanuary = firebase
  .database()
  .ref('days/2018')
  .orderByChild('dayOfYear')
  .limitToLast(10)
  .endAt(31);

var first10DaysOfJanuary = firebase
  .database()
  .ref('days/2018')
  .orderByChild('dayOfYear')
  .limitToFirst(100) // Limit is never hit
  .endAt(10); // endAt stops the query before it hits the limit
```

Authenticate Node.js

> Full admin privileges

```
var admin = require('firebase-admin');

// Fetch the service account key JSON file contents
var serviceAccount = require('path/to/serviceAccountKey.json');

// Initialize the app with a service account, granting admin privileges
admin.initializeApp({
  credential: admin.credential.cert(serviceAccount),
  databaseURL: 'https://databaseName.firebaseio.com',
});

// As an admin, the app has access to read and write all data, regardless of
Security Rules
var db = admin.database();
var ref = db.ref('restricted_access/secret_document');
ref.once('value', function(snapshot) {
  console.log(snapshot.val());
});
```

Initialize Node.js with limited privileges

> Set auth token variables to limit access

```
// Initialize the app with a custom auth variable, limiting the server's access
admin.initializeApp({
  credential: admin.credential.cert(serviceAccount),
  databaseURL: 'https://databaseName.firebaseio.com',
  databaseAuthVariableOverride: {
    uid: 'my-service-worker',
  },
});
```

> Act as an un-authenticated user

```
// Initialize the app with a custom auth variable, limiting the server's access
admin.initializeApp({
  credential: admin.credential.cert(serviceAccount),
  databaseURL: 'https://databaseName.firebaseio.com',
  databaseAuthVariableOverride: null,
});
```

Notes

Cloud Functions

See the Cloud Functions docs for Firebase.

Functions samples

See the official GitHub repo of Cloud Functions for Firebase sample functions

Mount an Express app

```
const functions = require('firebase-functions');
const express = require('express');
const cors = require('cors');
const app = express();

// Automatically allow cross-origin requests
app.use(cors({ origin: true }));

// Add middleware to authenticate requests
app.use(myMiddleware);

// build multiple CRUD interfaces:
app.get('/:id', (req, res) => res.send(Widgets.getById(req.params.id)));
app.post('/', (req, res) => res.send(Widgets.create()));
app.put('/:id', (req, res) => res.send(Widgets.update(req.params.id, req.body)));
app.delete('/:id', (req, res) => res.send(Widgets.delete(req.params.id)));
app.get('/', (req, res) => res.send(Widgets.list()));

// Expose Express API as a single Cloud Function:
exports.widgets = functions.https.onRequest(app);
```

Mount an Express handler

```
exports.helloWorld = functions.https.onRequest((req, res) => {
  res.status(200);
  res.send('hello world');
});
```

Firestore triggers

- onCreate
- onUpdate
- onDelete
- onWrite

```
exports.createUser = functions.firestore.document('users/{userId}').onCreate(event
=> {
  // Get an object representing the document
  // e.g. {'name': 'Marie', 'age': 66}
  var newValue = event.data.data();

  // access a particular field as you would any JS property
  var name = newValue.name;

  // perform desired operations ...
});
```

Realtime Database triggers

- onCreate
- onUpdate
- onDelete
- onWrite

```
exports.makeUppercase =
functions.database.ref('/messages/{pushId}/original').onWrite(event => {
  // Grab the current value of what was written to the Realtime Database.
  const original = event.data.val();
  console.log('Uppercasing', event.params.pushId, original);
  const uppercase = original.toUpperCase();
  // You must return a Promise when performing asynchronous tasks inside a
Functions such as
  // writing to the Firebase Realtime Database.
  // Setting an "uppercase" sibling in the Realtime Database returns a Promise.
  return event.data.ref.parent.child('uppercase').set(uppercase);
});
```

Firebase Authentication

- onCreate
- onDelete

```
exports.sendWelcomeEmail = functions.auth.user().onCreate(event => {
  const user = event.data; // The Firebase user.

  const email = user.email; // The email of the user.
  const displayName = user.displayName; // The display name of the user.
});
```

Firebase Storage

- onChange

```
exports.generateThumbnail = functions.storage.object().onChange(event => {
  const object = event.data; // The Storage object.

  const fileBucket = object.bucket; // The Storage bucket that contains the file.
  const filePath = object.name; // File path in the bucket.
  const contentType = object.contentType; // File content type.
  const resourceState = object.resourceState; // The resourceState is 'exists' or
'not_exists' (for file/folder deletions).
  const metageneration = object.metageneration; // Number of times metadata has
been generated. New objects have a value of 1.

  // Exit if this is triggered on a file that is not an image.
  if (!contentType.startsWith('image/')) {
    console.log('This is not an image.');
    return;
  }

  // Get the file name.
  const fileName = path.basename(filePath);
  // Exit if the image is already a thumbnail.
  if (fileName.startsWith('thumb_')) {
    console.log('Already a Thumbnail.');
    return;
  }

  // Exit if this is a move or deletion event.
  if (resourceState === 'not_exists') {
    console.log('This is a deletion event.');
    return;
  }

  // Exit if file exists but is not new and is only being triggered
  // because of a metadata change.
  if (resourceState === 'exists' && metageneration > 1) {
    console.log('This is a metadata change event.');
    return;
  }
});
```

Use ImageMagick

```
const functions = require('firebase-functions');
const gcs = require('@google-cloud/storage')();
const spawn = require('child-process-promise').spawn;
const path = require('path');
const os = require('os');
const fs = require('fs');

exports.generateThumbnail = functions.storage.object().onChange(event => {
  const object = event.data;

  const fileBucket = object.bucket;
  const filePath = object.name;
  const contentType = object.contentType;

  // Download file from bucket.
  const bucket = gcs.bucket(fileBucket);
  const tempFilePath = path.join(os.tmpdir(), fileName);
  const metadata = { contentType: contentType };
  return bucket
    .file(filePath)
    .download({
      destination: tempFilePath,
    })
    .then(() => {
      console.log('Image downloaded locally to', tempFilePath);
      // Generate a thumbnail using ImageMagick.
      return spawn('convert', [tempFilePath, '-thumbnail', '200x200>',
tempFilePath]);
    })
    .then(() => {
      console.log('Thumbnail created at', tempFilePath);
      // We add a 'thumb_' prefix to thumbnails file name. That's where we'll
upload the thumbnail.
      const thumbFileName = `thumb_${fileName}`;
      const thumbFilePath = path.join(path.dirname(filePath), thumbFileName);
      // Uploading the thumbnail.
      return bucket.upload(tempFilePath, { destination: thumbFilePath, metadata:
metadata });
      // Once the thumbnail has been uploaded delete the local file to free up
disk space.
    })
    .then(() => fs.unlinkSync(tempFilePath));
});
```

Notes

Firebase Storage

See the Firebase Storage docs for web.

Create a ref

```
var storageRef = firebase.storage().ref();
const fileRef = storageRef.child('/some/file/path.jpg);
```

Navigate

```
// Points to the root reference
var storageRef = firebase.storage().ref();

// Points to 'images'
var imagesRef = storageRef.child('images');

// Points to 'images/space.jpg'
// Note that you can use variables to create child values
var fileName = 'space.jpg';
var spaceRef = imagesRef.child(fileName);

// File path is 'images/space.jpg'
var path = spaceRef.fullPath;

// File name is 'space.jpg'
var name = spaceRef.name;

// Points to 'images'
var imagesRef = spaceRef.parent;
```

Upload file

```
// Create file metadata including the content type
var metadata = {
  contentType: 'image/jpeg',
};

// Upload the file and metadata
var uploadTask = storageRef.child('images/mountains.jpg').put(file, metadata);
```

Full example

```
function uploadFile(file) {
  // Create the file metadata
  var metadata = {
    contentType: 'image/jpeg',
  };

  // Upload file and metadata to the object 'images/mountains.jpg'
  var uploadTask = storageRef.child('images/' + file.name).put(file, metadata);

  // Listen for state changes, errors, and completion of the upload.
  uploadTask.on(
    firebase.storage.TaskEvent.STATE_CHANGED, // or 'state_changed'
    function(snapshot) {
      // Get task progress, including the number of bytes uploaded and the total
number of bytes to be uploaded
      var progress = snapshot.bytesTransferred / snapshot.totalBytes * 100;
      console.log('Upload is ' + progress + '% done');
      switch (snapshot.state) {
        case firebase.storage.TaskState.PAUSED: // or 'paused'
          console.log('Upload is paused');
          break;
        case firebase.storage.TaskState.RUNNING: // or 'running'
          console.log('Upload is running');
          break;
      }
    },
    function(error) {
      // Errors list: https://firebase.google.com/docs/storage/web/handle-errors
      switch (error.code) {
        case 'storage/unauthorized':
          // User doesn't have permission to access the object
          break;

        case 'storage/canceled':
          // User canceled the upload
          break;

        case 'storage/unknown':
          // Unknown error occurred, inspect error.serverResponse
          break;
      }
    },
    function() {
      // Upload completed successfully, now we can get the download URL
      var downloadURL = uploadTask.snapshot.downloadURL;
    }
  );
}
```

Download file

```
// Create a reference to the file we want to download
var starsRef = storageRef.child('images/stars.jpg');

// Get the download URL
starsRef.getDownloadURL().then(function(url) {
  // Insert url into an <img> tag to "download"
}).catch(function(error) {

  // A full list of error codes is available at
  // https://firebase.google.com/docs/storage/web/handle-errors
  switch (error.code) {
    case 'storage/object_not_found':
      // File doesn't exist
      break;

    case 'storage/unauthorized':
      // User doesn't have permission to access the object
      break;

    case 'storage/canceled':
      // User canceled the upload
      break;

    ...

    case 'storage/unknown':
      // Unknown error occurred, inspect the server response
      break;
  }
});
```

Set metadata

```
// Create a reference to the file whose metadata we want to change
var forestRef = storageRef.child('images/forest.jpg');

// Create file metadata to update
var newMetadata = {
  cacheControl: 'public,max-age=300',
  contentType: 'image/jpeg',
  contentLanguage: null,
  customMetadata: {
    whatever: 'we feel like',
  },
};

// Update metadata properties
forestRef
  .updateMetadata(newMetadata)
  .then(function(metadata) {
    // Updated metadata for 'images/forest.jpg' is returned in the Promise
  })
  .catch(function(error) {
    // Uh-oh, an error occurred!
  });
```

Notes

Cloud Messaging

See the Firebase Cloud Messaging docs for web.

manifest.json

```
{
  "gcm_sender_id": "103953800507"
}
```

Request permission in browser

```
// index.html
const messaging = firebase.messaging();
messaging
  .requestPermission()
  .then(function() {
    // Get Instance ID token. Initially this makes a network call, once retrieved
    // subsequent calls to getToken will return from cache.
    messaging.getToken()
    .then(function(currentToken) {
      if (currentToken) {
        sendTokenToServer(currentToken);
        updateUIForPushEnabled(currentToken);
      } else {
        // Show permission request.
        console.log('No Instance ID token available. Request permission to
generate one.');
        // Show permission UI.
        updateUIForPushPermissionRequired();
        setTokenSentToServer(false);
      }
    })
    .catch(function(err) {
      console.log('An error occurred while retrieving token. ', err);
      showToken('Error retrieving Instance ID token. ', err);
      setTokenSentToServer(false);
    });
  }
  .catch(function(err) {
    console.log('Unable to get permission to notify.', err);
  });
```

Monitor token refresh

```
// index.html
// Callback fired if Instance ID token is updated.
messaging.onTokenRefresh(function() {
  messaging
    .getToken()
    .then(function(refreshedToken) {
      console.log('Token refreshed.');
      // Indicate that the new Instance ID token has not yet been sent to the
      // app server.
      setTokenSentToServer(false);
      // Send Instance ID token to app server.
      sendTokenToServer(refreshedToken);
      // ...
    })
    .catch(function(err) {
      console.log('Unable to retrieve refreshed token ', err);
      showToken('Unable to retrieve refreshed token ', err);
    });
});
```

Catch messages when page is in foreground

```
// index.html
// Handle incoming messages. Called when:
// - a message is received while the app has focus
// - the user clicks on an app notification created by a sevice worker
//   `messaging.setBackgroundMessageHandler` handler.
messaging.onMessage(function(payload) {
  console.log('Message received. ', payload);
  // ...
});
```

Create serviceWorker

> You need a serviceWorker to listen for messages in the background

```
// firebase-messaging-sw.js
// Give the service worker access to Firebase Messaging.
// Note that you can only use Firebase Messaging here, other Firebase libraries
// are not available in the service worker.
importScripts('https://www.gstatic.com/firebasejs/4.8.1/firebase-app.js');
importScripts('https://www.gstatic.com/firebasejs/4.8.1/firebase-messaging.js');

// Initialize the Firebase app in the service worker by passing in the
// messagingSenderId.
firebase.initializeApp({
  messagingSenderId: 'YOUR-SENDER-ID',
});

// Retrieve an instance of Firebase Messaging so that it can handle background
// messages.
const messaging = firebase.messaging();
```

Send message to single recipient

```
// Cloud Function
// This registration token comes from the client FCM SDKs.
var registrationToken = 'bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...';

// See the "Defining the message payload" section below for details
// on how to define a message payload.
var payload = {
  notification: {
    title: 'Title of your push notification',
    body: 'Body of your push notification',
    click_action: 'https://dummypage.com',
  },
  data: {
    score: '850',
    time: '2:45',
  },
};

// Send a message to the device corresponding to the provided
// registration token.
admin
  .messaging()
  .sendToDevice(registrationToken, payload)
  .then(function(response) {
    // See the MessagingDevicesResponse reference documentation for
    // the contents of response.
    console.log('Successfully sent message:', response);
  })
  .catch(function(error) {
    console.log('Error sending message:', error);
  });
```

Send multi-cast message

```
// Cloud Function
// These registration tokens come from the client FCM SDKs.
var registrationTokens = [
  'bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...',
  // ...
  'ecupwIfBy1w:APA91bFtuMY7MktgxA3Au_Qx7cKqnf...',
];

//...
admin
  .messaging()
  .sendToDevice(registrationTokens, payload)
  .then(function(response) {
    //...
  });
```

Send device-group message

> See managing device groups

```
// Cloud Function
var notificationKey = 'some-notification-key';

//...
admin
  .messaging()
  .sendToDeviceGroup(notificationKey, payload)
  .then(function(response) {
    // ...
  });
```

Send topic message

> See managing device groups

```
// Cloud Function
// The topic name can be optionally prefixed with "/topics/".
var topic = 'highScores';

//...

admin
  .messaging()
  .sendToTopic(topic, payload)
  .then(function(response) {
    //...
  });
```

Send to condition

> Conditions support only two operations per expression

```
// Cloud Function
// Define a condition which will send to devices which are subscribed
// to either the Google stock or the tech industry topics.
var condition = "'stock-GOOG' in topics || 'industry-tech' in topics";

//...
admin
  .messaging()
  .sendToCondition(condition, payload)
  .then(function(response) {
    //...
  });
```

Message options

```
// Cloud Function
var registrationToken = 'bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...';

var payload = {
  notification: {
    title: 'Urgent action needed!',
    body: 'Urgent action is needed to prevent your account from being disabled!',
  },
};

// Set the message as high priority and have it expire after 24 hours.
var options = {
  priority: 'high',
  timeToLive: 60 * 60 * 24,
};

admin
  .messaging()
  .sendToDevice(registrationToken, payload, options)
  .then(function(response) {
    //...
  });
```

Subscribe to topic

```
// Cloud Function
var registrationTokens = [
  'bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...',
  // ...
  'ecupwIfBy1w:APA91bFtuMY7MktgxA3Au_Qx7cKqnf...',
];

admin
  .messaging()
  .subscribeToTopic(registrationTokens, topic)
  .then(function(response) {
    //...
  });
```

Subscribe to topic

```
// Cloud Function
const registrationTokens = [
  'bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...',
  // ...
  'ecupwIfBy1w:APA91bFtuMY7MktgxA3Au_Qx7cKqnf...',
];

const topic = 'highScores';

admin
  .messaging()
  .subscribeToTopic(registrationTokens, topic)
  .then(function(response) {
    //...
  });
```

Unsubscribe to topic

```
// Cloud Function
const registrationTokens = [
  'bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...',
  // ...
  'ecupwIfBy1w:APA91bFtuMY7MktgxA3Au_Qx7cKqnf...',
];

const topic = 'highScores';

admin
  .messaging()
  .unsubscribeFromTopic(registrationTokens, topic)
  .then(function(response) {
    //...
  });
```

Notes

Firebase Hosting

See the Firebase Hosting doc for web.

Redirects

```
"hosting": {
  // Add the "redirects" section within "hosting"
  "redirects": [ {
    "source" : "/foo",
    "destination" : "/bar",
    "type" : 301
  }, {
    "source" : "/firebase/*",
    "destination" : "https://firebase.google.com",
    "type" : 302
  } ]
}
```

Rewrites

```
"hosting": {
  // Add the "rewrites" section within "hosting"
  "rewrites": [ {
    "source": "**",
    "destination": "/index.html"
  } ]
}
```

Headers

```
"hosting": {
    // Add the "headers" section within "hosting".
    "headers": [ {
      "source" : "**/*.@(eot|otf|ttf|ttc|woff|font.css)",
      "headers" : [ {
        "key" : "Access-Control-Allow-Origin",
        "value" : "*"
    } ]
    }, {
      "source" : "**/*.@(jpg|jpeg|gif|png)",
      "headers" : [ {
      "key" : "Cache-Control",
      "value" : "max-age=7200"
      } ]
    }, {
      // Sets the cache header for 404 pages to cache for 5 minutes
      "source" : "404.html",
      "headers" : [ {
      "key" : "Cache-Control",
      "value" : "max-age=300"
      } ]
    } ]
 }
```

Connect a Cloud Function

```
{
  "hosting": {
    "public": "public",

    // Add the following rewrites section *within* "hosting"
   "rewrites": [ {
      "source": "/bigben", "function": "bigben"
    } ]

  }
}
```