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');
```