JSONStore Code Examples

Cordova

Initialize and open connections, get an Accessor, and add data

```
var collectionName = 'people';

// Object that defines all the collections.
var collections = {

  // Object that defines the 'people' collection.
  people : {

    // Object that defines the Search Fields for the 'people' collection.
    searchFields : {name: 'string', age: 'integer'}
  }
};

// Optional options object.
var options = {

  // Optional username, default 'jsonstore'.
  username : 'carlos',

  // Optional password, default no password.
  password : '123',

  // Optional local key generation flag, default false.
  localKeyGen : false
};

WL.JSONStore.init(collections, options)

.then(function () {

  // Data to add, you probably want to get
  // this data from a network call (e.g. MobileFirst Adapter).
  var data = [{name: 'carlos', age: 10}];

  // Optional options for add.
  var addOptions = {

    // Mark data as dirty (true = yes, false = no), default true.
    markDirty: true
  };

  // Get an accessor to the people collection and add data.
  return WL.JSONStore.get(collectionName).add(data, addOptions);
})

.then(function (numberOfDocumentsAdded) {
  // Add was successful.
})

.fail(function (errorObject) {
   // Handle failure for any of the previous JSONStore operations (init, add).
});
```

Find - locate documents inside the Store

```
var collectionName = 'people';

// Find all documents that match the queries.
var queryPart1 = WL.JSONStore.QueryPart()
                  .equal('name', 'carlos')
                  .lessOrEqualThan('age', 10)

var options = {
  // Returns a maximum of 10 documents, default no limit.
  limit: 10,

  // Skip 0 documents, default no offset.
  offset: 0,

  // Search fields to return, default: ['_id', 'json'].
  filter: ['_id', 'json'],

  // How to sort the returned values, default no sort.
  sort: [{name: WL.constant.ASCENDING}, {age: WL.constant.DESCENDING}]
};

WL.JSONStore.get(collectionName)

// Alternatives:
// - findById(1, options) which locates documents by their _id field
// - findAll(options) which returns all documents
// - find({'name': 'carlos', age: 10}, options) which finds all documents
// that match the query.
.advancedFind([queryPart1], options)

.then(function (arrayResults) {
  // arrayResults = [{_id: 1, json: {name: 'carlos', age: 99}}]
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Replace - change the documents that are already stored inside a Collection

```
var collectionName = 'people';

// Documents will be located with their '_id' field
// and replaced with the data in the 'json' field.
var docs = [{_id: 1, json: {name: 'carlitos', age: 99}}];

var options = {

  // Mark data as dirty (true = yes, false = no), default true.
  markDirty: true
};

WL.JSONStore.get(collectionName)

.replace(docs, options)

.then(function (numberOfDocumentsReplaced) {
  // Handle success.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Remove - delete all documents that match the query

```
var collectionName = 'people';

// Remove all documents that match the queries.
var queries = [{_id: 1}];

var options = {

  // Exact match (true) or fuzzy search (false), default fuzzy search.
  exact: true,

  // Mark data as dirty (true = yes, false = no), default true.
  markDirty: true
};

WL.JSONStore.get(collectionName)

.remove(queries, options)

.then(function (numberOfDocumentsRemoved) {
  // Handle success.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Count - gets the total number of documents that match a query

```
var collectionName = 'people';

// Count all documents that match the query.
// The default query is '{}' which will
// count every document in the collection.
var query = {name: 'carlos'};
var options = {

  // Exact match (true) or fuzzy search (false), default fuzzy search.
  exact: true
};

WL.JSONStore.get(collectionName)

.count(query, options)

.then(function (numberOfDocumentsThatMatchedTheQuery) {
  // Handle success.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Destroy - wipes data for all users, destroys the internal storage, and clears security artifacts

```
WL.JSONStore.destroy()

.then(function () {
  // Handle success.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Security - close access to all opened Collections for the current user

```
WL.JSONStore.closeAll()

.then(function () {
  // Handle success.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Security - change the password that is used to access a Store

```
// The password should be user input.
// It is hard-coded in the example for brevity.
var oldPassword = '123';
var newPassword = '456';

var clearPasswords = function () {
  oldPassword = null;
  newPassword = null;
};

// Default username if none is passed is: 'jsonstore'.
var username = 'carlos';

WL.JSONStore.changePassword(oldPassword, newPassword, username)

.then(function () {

  // Make sure you do not leave the password(s) in memory.
  clearPasswords();

  // Handle success.
})

.fail(function (errorObject) {

  // Make sure you do not leave the password(s) in memory.
  clearPasswords();

  // Handle failure.
});
```

Push - get all documents that are marked as dirty, send them to a MobileFirst adapter, and mark them clean

```
var collectionName = 'people';
var dirtyDocs;

WL.JSONStore.get(collectionName)

.getAllDirty()

.then(function (arrayOfDirtyDocuments) {
  // Handle getAllDirty success.

  dirtyDocs = arrayOfDirtyDocuments;

  var procedure = 'procedure-name-1';
  var adapter = 'adapter-name';

  var resource = new WLResourceRequest("adapters/" + adapter + "/" + procedure, W
LResourceRequest.GET);
  resource.setQueryParameter('params', [dirtyDocs]);
  return resource.send();
})

.then(function (responseFromAdapter) {
  // Handle invokeProcedure success.

  // You may want to check the response from the adapter
  // and decide whether or not to mark documents as clean.
  return WL.JSONStore.get(collectionName).markClean(dirtyDocs);
})

.then(function () {
  // Handle markClean success.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Pull - get new data from a MobileFirst adapter

```
var collectionName = 'people';

var adapter = 'adapter-name';
var procedure = 'procedure-name-2';

var resource = new WLResourceRequest("adapters/" + adapter + "/" + procedure, WLR
esourceRequest.GET);

resource.send()

.then(function (responseFromAdapter) {
  // Handle invokeProcedure success.

  // The following example assumes that the adapter returns an arrayOfData,
  // (which is not returned by default),
  // as part of the invocationResult object,
  // with the data that you want to add to the collection.
  var data = responseFromAdapter.responseJSON

  // Example:
  // data = [{id: 1, ssn: '111-22-3333', name: 'carlos'}];

  var changeOptions = {

    // The following example assumes that 'id' and 'ssn' are search fields,
    // default will use all search fields
    // and are part of the data that is received.
    replaceCriteria : ['id', 'ssn'],

    // Data that does not exist in the Collection will be added, default false.
    addNew : true,

    // Mark data as dirty (true = yes, false = no), default false.
    markDirty : false
  };

  return WL.JSONStore.get(collectionName).change(data, changeOptions);
})

.then(function () {
  // Handle change success.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Check whether a document is dirty

```
var collectionName = 'people';
var doc = {_id: 1, json: {name: 'carlitos', age: 99}};

WL.JSONStore.get(collectionName)

.isDirty(doc)

.then(function (isDocumentDirty) {
  // Handle success.

  // isDocumentDirty - true if dirty, false otherwise.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Check the number of dirty documents

```
var collectionName = 'people';

WL.JSONStore.get(collectionName)

.countAllDirty()

.then(function (numberOfDirtyDocuments) {
  // Handle success.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Remove a Collection

```
var collectionName = 'people';

WL.JSONStore.get(collectionName)

.removeCollection()

.then(function () {
  // Handle success.

  // Note: You must call the 'init' API to re-use the empty collection.
  // See the 'clear' API if you just want to remove all data that is inside.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Clear all data that is inside a Collection

```
var collectionName = 'people';

WL.JSONStore.get(collectionName)

.clear()

.then(function () {
  // Handle success.

  // Note: You might want to use the 'removeCollection' API
  // instead if you want to change the search fields.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Start a transaction, add some data, remove a document, commit the transaction and roll back the transaction if there is a failure

```
WL.JSONStore.startTransaction()

.then(function () {
  // Handle startTransaction success.
  // You can call every JSONStore API method except:
  // init, destroy, removeCollection, and closeAll.

  var data = [{name: 'carlos'}];

  return WL.JSONStore.get(collectionName).add(data);
})

.then(function () {

  var docs = [{_id: 1, json: {name: 'carlos'}}];

  return WL.JSONStore.get(collectionName).remove(docs);
})

.then(function () {

  return WL.JSONStore.commitTransaction();
})

.fail(function (errorObject) {
  // Handle failure for any of the previous JSONStore operation.
  //(startTransaction, add, remove).

  WL.JSONStore.rollbackTransaction()

  .then(function () {
    // Handle rollback success.
  })

  .fail(function () {
    // Handle rollback failure.
  })

});
```

Get file information

```
WL.JSONStore.fileInfo()
.then(function (res) {
  //res => [{isEncrypted : true, name : carlos, size : 3072}]
})

  .fail(function () {
  // Handle failure.
});
```

Search with like, rightLike, and leftLike

```
// Match all records that contain the search string on both sides.
// %searchString%
var arr1 = WL.JSONStore.QueryPart().like('name', 'ca');  // returns {name: 'carlo
s', age: 10}
var arr2 = WL.JSONStore.QueryPart().like('name', 'los');  // returns {name: 'carl
os', age: 10}

// Match all records that contain the search string on the left side and anything
on the right side.
// searchString%
var arr1 = WL.JSONStore.QueryPart().rightLike('name', 'ca');  // returns {name: '
carlos', age: 10}
var arr2 = WL.JSONStore.QueryPart().rightLike('name', 'los');  // returns nothing

// Match all records that contain the search string on the right side and anythin
g on the left side.
// %searchString
var arr = WL.JSONStore.QueryPart().leftLike('name', 'ca');  // returns nothing
var arr2 = WL.JSONStore.QueryPart().leftLike('name', 'los');  // returns {name: '
carlos', age: 10}
```

iOS

Initialize and open connections, get an Accessor, and add data

```
// Create the collections object that will be initialized.
JSONStoreCollection* people = [[JSONStoreCollection alloc] initWithName:@"peop
le"];
[people setSearchField:@"name" withType:JSONStore_String];
[people setSearchField:@"age" withType:JSONStore_Integer];

// Optional options object.
JSONStoreOpenOptions* options = [JSONStoreOpenOptions new];
[options setUsername:@"carlos"]; //Optional username, default 'jsonstore'
[options setPassword:@"123"]; //Optional password, default no password

// This object will point to an error if one occurs.
NSError* error = nil;

// Open the collections.
[[JSONStore sharedInstance] openCollections:@[people] withOptions:options error:&
error];

// Add data to the collection
NSArray* data = @[ @{@"name" : @"carlos", @"age": @10} ];
int newDocsAdded = [[people addData:data andMarkDirty:YES withOptions:nil error:&
error] intValue];
Initialize with a secure random token from the server
[WLSecurityUtils getRandomStringFromServerWithBytes:32
                 timeout:1000
                 completionHandler:^(NSURLResponse *response,
                                     NSData *data,
                                     NSError *connectionError) {

  // You might want to see the response and the connection error
  // before moving forward
```

```
  // before moving forward.

  // Get the secure random string by using the data that is
  // returned from the generator on the server.
  NSString* secureRandom = [[NSString alloc] initWithData:data encoding:NSUTF8Str
ingEncoding];

  JSONStoreCollection* ppl = [[JSONStoreCollection alloc] initWithName:@"people"]
;
  [ppl setSearchField:@"name" withType:JSONStore_String];
  [ppl setSearchField:@"age" withType:JSONStore_Integer];

  // Optional options object.
  JSONStoreOptions* options = [JSONStoreOptions new];
  [options setUsername:@"carlos"]; //Optional username, default 'jsonstore'
  [options setPassword:@"123"]; //Optional password, default no password
  [options setSecureRandom:secureRandom]; //Optional, default one will be generat
ed locally

  // This points to an error if one occurs.
  NSError* error = nil;

  [[JSONStore sharedInstance] openCollections:@[ppl] withOptions:options error:&e
rror];

  // Other JSONStore operations (e.g. add, remove, replace, etc.) go here.
}];
```

Find - locate documents inside the Store

```
// Get the accessor to an already initialized collection.
JSONStoreCollection* people = [[JSONStore sharedInstance] getCollectionWithNam
e:@"people"];

// This object will point to an error if one occurs.
NSError* error = nil;

// Add additional find options (optional).
JSONStoreQueryOptions* options = [JSONStoreQueryOptions new];
[options setLimit:@10]; // Returns a maximum of 10 documents, default no limit.
[options setOffset:@0]; // Skip 0 documents, default no offset.

// Search fields to return, default: ['_id', 'json'].
[options filterSearchField:@"_id"];
[options filterSearchField:@"json"];

// How to sort the returned values , default no sort.
[options sortBySearchFieldAscending:@"name"];
[options sortBySearchFieldDescending:@"age"];

// Find all documents that match the query part.
JSONStoreQueryPart* queryPart1 = [[JSONStoreQueryPart alloc] init];
[queryPart1 searchField:@"name" equal:@"carlos"];
[queryPart1 searchField:@"age" lessOrEqualThan:@10];

NSArray* results = [people findWithQueryParts:@[queryPart1] andOptions:options er
ror:&error];

// results = @[ @{@"_id" : @1, @"json" : @{ @"name": @"carlos", @"age" : @10}} ];

for (NSDictionary* result in results) {

  NSString* name = [result valueForKeyPath:@"json.name"]; // carlos.
  int age = [[result valueForKeyPath:@"json.age"] intValue]; // 10
  NSLog(@"Name: %@, Age: %d", name, age);
}
```

Replace - change the documents that are already stored inside a Collection

```
// Get the accessor to an already initialized collection.
JSONStoreCollection* people = [[JSONStore sharedInstance] getCollectionWithNam
e:@"people"];

// Find all documents that match the queries.
NSArray* docs = @[ @{@"_id" : @1, @"json" : @{ @"name": @"carlitos", @"age" : @99
}} ];


// This object will point to an error if one occurs.
NSError* error = nil;

// Perform the replacement.
int docsReplaced = [[people replaceDocuments:docs andMarkDirty:NO error:&error] i
ntValue];
```

Remove - delete all documents that match the query

```
// Get the accessor to an already initialized collection.
JSONStoreCollection* people = [[JSONStore sharedInstance] getCollectionWithNam
e:@"people"];

// This object will point to an error if one occurs.
NSError* error = nil;

// Find document with _id equal to 1 and remove it.
int docsRemoved = [[people removeWithIds:@[@1] andMarkDirty:NO error:&error] intV
alue];
```

Count - gets the total number of documents that match a query

```
// Get the accessor to an already initialized collection.
JSONStoreCollection* people = [[JSONStore sharedInstance] getCollectionWithNam
e:@"people"];

// Count all documents that match the query.
// The default query is @{} which will
// count every document in the collection.
JSONStoreQueryPart *queryPart = [[JSONStoreQueryPart alloc] init];
[queryPart searchField:@"name" equal:@"carlos"];

// This object will point to an error if one occurs.
NSError* error = nil;

// Perform the count.
int countResult = [[people countWithQueryParts:@[queryPart] error:&error] intValu
e];
```

Destroy - wipes data for all users, destroys the internal storage, and clears security artifacts

```
// This object will point to an error if one occurs.
NSError* error = nil;

// Perform the destroy.
[[JSONStore sharedInstance] destroyDataAndReturnError:&error];
```

Security - close access to all opened Collections for the current user

```
// This object will point to an error if one occurs.
NSError* error = nil;

// Close access to all collections in the store.
[[JSONStore sharedInstance] closeAllCollectionsAndReturnError:&error];
```

Security - change the password that is used to access a Store

```
// The password should be user input.
// It is hardcoded in the example for brevity.
NSString* oldPassword = @"123";
NSString* newPassword = @"456";
NSString* username = @"carlos";

// This object will point to an error if one occurs.
NSError* error = nil;

// Perform the change password operation.
[[JSONStore sharedInstance] changeCurrentPassword:oldPassword withNewPassword:new
Password forUsername:username error:&error];

// Remove the passwords from memory.
oldPassword = nil;
newPassword = nil;
```

Push - get all documents that are marked as dirty, send them to a MobileFirst adapter, and mark them clean

```
// Get the accessor to an already initialized collection.
JSONStoreCollection* people = [[JSONStore sharedInstance] getCollectionWithNam
e:@"people"];

// This object will point to an error if one occurs
NSError* error = nil;

// Return all documents marked dirty
NSArray* dirtyDocs = [people allDirtyAndReturnError:&error];

// ACTION REQUIRED: Handle the dirty documents here
// (e.g. send them to a MobileFirst Adapter).

// Mark dirty documents as clean
int numCleaned = [[people markDocumentsClean:dirtyDocs error:&error] intValue];
```

Pull - get new data from a MobileFirst adapter

```
// Get the accessor to an already initialized collection.
JSONStoreCollection* people = [[JSONStore sharedInstance] getCollectionWithNam
e:@"people"];

// This object will point to an error if one occurs.
NSError* error = nil;


// ACTION REQUIRED: Get data (e.g. MobileFirst Adapter).
// For this example, it is hardcoded.
NSArray* data = @[ @{@"id" : @1, @"ssn": @"111-22-3333", @"name": @"carlos"} ];


int numChanged = [[people changeData:data withReplaceCriteria:@[@"id", @"ssn"] ad
dNew:YES markDirty:NO error:&error] intValue];
```

Check whether a document is dirty

```
// Get the accessor to an already initialized collection.
JSONStoreCollection* people = [[JSONStore sharedInstance] getCollectionWithNam
e:@"people"];

// This object will point to an error if one occurs.
NSError* error = nil;

// Check if document with _id '1' is dirty.
BOOL isDirtyResult = [people isDirtyWithDocumentId:1 error:&error];
```

Check the number of dirty documents

```
// Get the accessor to an already initialized collection.
JSONStoreCollection* people = [[JSONStore sharedInstance] getCollectionWithNam
e:@"people"];

// This object will point to an error if one occurs.
NSError* error = nil;

// Check if document with _id '1' is dirty.
int dirtyDocsCount = [[people countAllDirtyDocumentsWithError:&error] intValue];
```

Remove a Collection

```
// Get the accessor to an already initialized collection.
JSONStoreCollection* people = [[JSONStore sharedInstance] getCollectionWithNam
e:@"people"];

// This object will point to an error if one occurs.
NSError* error = nil;

// Remove the collection.
[people removeCollectionWithError:&error];
```

Clear all data that is inside a Collection

```
// Get the accessor to an already initialized collection.
JSONStoreCollection* people = [[JSONStore sharedInstance] getCollectionWithNam
e:@"people"];

// This object will point to an error if one occurs.
NSError* error = nil;

// Remove the collection.
[people clearCollectionWithError:&error];
```

Start a transaction, add some data, remove a document, commit the transaction and roll back the transaction if there is a failure

```
// Get the accessor to an already initialized collection.
JSONStoreCollection* people = [[JSONStore sharedInstance] getCollectionWithNam
e:@"people"];

// These objects will point to errors if they occur.
NSError* error = nil;
NSError* addError = nil;
NSError* removeError = nil;

// You can call every JSONStore API method inside a transaction except:
// open, destroy, removeCollection and closeAll.
[[JSONStore sharedInstance] startTransactionAndReturnError:&error];

[people addData:@[ @{@"name" : @"carlos"} ] andMarkDirty:NO withOptions:nil error
:&addError];

[people removeWithIds:@[@1] andMarkDirty:NO error:&removeError];

if (addError != nil || removeError != nil) {

  // Return the store to the state before start transaction was called.
  [[JSONStore sharedInstance] rollbackTransactionAndReturnError:&error];
} else {
  // Commit the transaction thus ensuring atomicity.
  [[JSONStore sharedInstance] commitTransactionAndReturnError:&error];
}
```

Get file information

```
// This object will point to an error if one occurs
NSError* error = nil;

// Returns information about files JSONStore uses to persist data.
NSArray* results = [[JSONStore sharedInstance] fileInfoAndReturnError:&error];
// => [{@"isEncrypted" : @(true), @"name" : @"carlos", @"size" : @3072}]
```

Android

Initialize and open connections, get an Accessor, and add data

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  List<JSONStoreCollection> collections = new LinkedList<JSONStoreCollection>();
  // Create the collections object that will be initialized.
  JSONStoreCollection peopleCollection = new JSONStoreCollection("people");
  peopleCollection.setSearchField("name", SearchFieldType.STRING);
  peopleCollection.setSearchField("age", SearchFieldType.INTEGER);
  collections.add(peopleCollection);

  // Optional options object.
  JSONStoreInitOptions initOptions = new JSONStoreInitOptions();
  // Optional username, default 'jsonstore'.
  initOptions.setUsername("carlos");
  // Optional password, default no password.
  initOptions.setPassword("123");

  // Open the collection.

  WLJSONStore.getInstance(ctx).openCollections(collections, initOptions);

  // Add data to the collection.
  JSONObject newDocument = new JSONObject("{name: 'carlos', age: 10}");
  JSONStoreAddOptions addOptions = new JSONStoreAddOptions();
  addOptions.setMarkDirty(true);
  peopleCollection.addData(newDocument, addOptions);
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations (init, add).
  throw ex;
} catch (JSONException ex) {
  // Handle failure for any JSON parsing issues.
throw ex;
}
```

Initialize with a secure random token from the server

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

// Do an AsyncTask because networking cannot occur inside the activity.
AsyncTask<Context, Void, Void> aTask = new AsyncTask<Context, Void, Void>() {
  protected Void doInBackground(Context... params) {
    final Context context = params[0];

    // Create the request listener that will have the
    // onSuccess and onFailure callbacks:
    WLRequestListener listener = new WLRequestListener() {
      public void onFailure(WLFailResponse failureResponse) {
        // Handle Failure.
      }

      public void onSuccess(WLResponse response) {
        String secureRandom = response.getResponseText();
```

```
        try {
          List<JSONStoreCollection> collections = new LinkedList<JSONStoreCollect
ion>();
          // Create the collections object that will be initialized.
          JSONStoreCollection peopleCollection = new JSONStoreCollection("people"
);
          peopleCollection.setSearchField("name", SearchFieldType.STRING);
          peopleCollection.setSearchField("age", SearchFieldType.INTEGER);
          collections.add(peopleCollection);

          // Optional options object.
          JSONStoreInitOptions initOptions = new JSONStoreInitOptions();

          // Optional username, default 'jsonstore'.
          initOptions.setUsername("carlos");

          // Optional password, default no password.
          initOptions.setPassword("123");

          initOptions.setSecureRandom(secureRandom);

          // Open the collection.
          WLJSONStore.getInstance(context).openCollections(collections, initOptio
ns);

          // Other JSONStore operations (e.g. add, remove, replace, etc.) go here
.
        }
        catch (JSONStoreException ex) {
          // Handle failure for any of the previous JSONStore operations (init, a
dd).
          ex.printStackTrace();        }
      }
    };

    // Get the secure random from the server:
    // The length of the random string, in bytes (maximum is 64 bytes).
    int byteLength = 32;
    SecurityUtils.getRandomStringFromServer(byteLength, context, listener);
    return null;
  }
};
aTask.execute(ctx);
```

Find - locate documents inside the Store

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollect
ionByName("people");

  JSONStoreQueryParts findQuery = new JSONStoreQueryParts();
  JSONStoreQueryPart part = new JSONStoreQueryPart();
  part.addLike("name", "carlos");
  part.addLessThan("age", 99);
  findQuery.addQueryPart(part);

  // Add additional find options (optional).
  JSONStoreFindOptions findOptions = new JSONStoreFindOptions();

  // Returns a maximum of 10 documents, default no limit.
  findOptions.setLimit(10);
  // Skip 0 documents, default no offset.
  findOptions.setOffset(0);

  // Search fields to return, default: ['_id', 'json'].
  findOptions.addSearchFilter("_id");
  findOptions.addSearchFilter("json");

  // How to sort the returned values, default no sort.
  findOptions.sortBySearchFieldAscending("name");
  findOptions.sortBySeachFieldDescending("age");

  // Find documents that match the query.
  List<JSONObject> results = peopleCollection.findDocuments(findQuery, findOption
s);
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations
  throw ex;
}
```

Replace - change the documents that are already stored inside a Collection

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollect
ionByName("people");

  // Documents will be located with their '_id' field
  //and replaced with the data in the 'json' field.
  JSONObject replaceDoc = new JSONObject("{_id: 1, json: {name: 'carlitos', age:
99}}");

  // Mark data as dirty (true = yes, false = no), default true.
  JSONStoreReplaceOptions replaceOptions = new JSONStoreReplaceOptions();
  replaceOptions.setMarkDirty(true);

  // Replace the document.
  peopleCollection.replaceDocument(replaceDoc, replaceOptions);
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.
  throw ex;
}
```

Remove - delete all documents that match the query

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollect
ionByName("people");

  // Documents will be located with their '_id' field.
  int id = 1;

  JSONStoreRemoveOptions removeOptions = new JSONStoreRemoveOptions();

  // Mark data as dirty (true = yes, false = no), default true.
  removeOptions.setMarkDirty(true);

  // Replace the document.
  peopleCollection.removeDocumentById(id, removeOptions);
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations
  throw ex;
}
catch (JSONException ex) {
  // Handle failure for any JSON parsing issues.
  throw ex;
}
```

Count - gets the total number of documents that match a query

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollect
ionByName("people");

  // Count all documents that match the query.
  JSONStoreQueryParts countQuery = new JSONStoreQueryParts();
  JSONStoreQueryPart part = new JSONStoreQueryPart();

  // Exact match.
  part.addEqual("name", "carlos");
  countQuery.addQueryPart(part);

  // Replace the document.
  int resultCount = peopleCollection.countDocuments(countQuery);
  JSONObject doc = peopleCollection.findDocumentById(resultCount);
  peopleCollection.replaceDocument(doc);
}
catch (JSONStoreException ex) {
  throw ex;
}
```

Destroy - wipes data for all users, destroys the internal storage, and clears security artifacts

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Destroy the Store.
  WLJSONStore.getInstance(ctx).destroy();
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations
  throw ex;
}
```

Security - close access to all opened Collections for the current user

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Close access to all collections.
  WLJSONStore.getInstance(ctx).closeAll();
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.
  throw ex;
}
```

Security - change the password that is used to access a Store

```
// The password should be user input.
// It is hard-coded in the example for brevity.
String username = "carlos";
String oldPassword = "123";
String newPassword = "456";

// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  WLJSONStore.getInstance(ctx).changePassword(oldPassword, newPassword, username)
;
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.
  throw ex;
}
finally {
  // It is good practice to not leave passwords in memory
  oldPassword = null;
  newPassword = null;
}
```

Push - get all documents that are marked as dirty, send them to a MobileFirst adapter, and mark them clean

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollect
ionByName("people");

  // Check if document with _id 3 is dirty.
  List<JSONObject> allDirtyDocuments = peopleCollection.findAllDirtyDocuments();

  // Handle the dirty documents here (e.g. calling an adapter).

  peopleCollection.markDocumentsClean(allDirtyDocuments);
}  catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations
  throw ex;
}
```

Pull - get new data from a MobileFirst adapter

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollect
ionByName("people");

  // Pull data here and place in newDocs. For this example, it is hard-coded.
  List<JSONObject> newDocs = new ArrayList<JSONObject>();
  JSONObject doc = new JSONObject("{id: 1, ssn: '111-22-3333', name: 'carlos'}");
  newDocs.add(doc);

  JSONStoreChangeOptions changeOptions = new JSONStoreChangeOptions();

  // Data that does not exist in the collection will be added, default false.
  changeOptions.setAddNew(true);

  // Mark data as dirty (true = yes, false = no), default false.
  changeOptions.setMarkDirty(true);

  // The following example assumes that 'id' and 'ssn' are search fields,
  // default will use all search fields
  // and are part of the data that is received.
  changeOptions.addSearchFieldToCriteria("id");
  changeOptions.addSearchFieldToCriteria("ssn");

  int changed = peopleCollection.changeData(newDocs, changeOptions);
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.
  throw ex;
}
catch (JSONException ex) {
  // Handle failure for any JSON parsing issues.
  throw ex;
}
```

Check whether a document is dirty

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollect
ionByName("people");

  // Check if document with id '3' is dirty.
  boolean isDirty = peopleCollection.isDocumentDirty(3);
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.
  throw ex;
}
```

Check the number of dirty documents

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollect
ionByName("people");

  // Get the count of all dirty documents in the people collection.
  int totalDirty = peopleCollection.countAllDirtyDocuments();
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.
  throw ex;
}
```

Remove a Collection

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollect
ionByName("people");

  // Remove the collection. The collection object is
  // no longer usable.
  peopleCollection.removeCollection();
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.
  throw ex;
}
```

Clear all data that is inside a Collection

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollect
ionByName("people");

  // Clear the collection.
  peopleCollection.clearCollection();
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.
  throw ex;
}
```

Start a transaction, add some data, remove a document, commit the transaction and roll back the transaction if there is a failure

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollect
ionByName("people");

  WLJSONStore.getInstance(ctx).startTransaction();

  JSONObject docToAdd = new JSONObject("{name: 'carlos', age: 99}");
  // Find documents that match query.
  peopleCollection.addData(docToAdd);


  //Remove added doc.
  int id = 1;
  peopleCollection.removeDocumentById(id);

  WLJSONStore.getInstance(ctx).commitTransaction();
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.

  // An exception occured. Take care of it to prevent further damage.
  WLJSONStore.getInstance(ctx).rollbackTransaction();

  throw ex;
}
catch (JSONException ex) {
  // Handle failure for any JSON parsing issues.

  // An exception occured. Take care of it to prevent further damage.
  WLJSONStore.getInstance(ctx).rollbackTransaction();

  throw ex;
}
```

Get file information

```
Context ctx = getContext();
List<JSONStoreFileInfo> allFileInfo = WLJSONStore.getInstance(ctx).getFileInfo
();

for(JSONStoreFileInfo fileInfo : allFileInfo) {
  long fileSize = fileInfo.getFileSizeBytes();
  String username = fileInfo.getUsername();
  boolean isEncrypted = fileInfo.isEncrypted();
}
```