

Using JSONStore in Native Android applications

Overview

This tutorial is a continuation of the JSONStore Overview tutorial.

The tutorial covers the following topics:

- Basic API Usage
- Advanced Usage
- Sample application
- Additional information

Basic API Usage

Open

Use `openCollections` to open one or more JSONStore collections

Starting or provisioning a collections means creating the persistent storage that contains the collection and documents, if it does not exists.

If the persistent storage is encrypted and a correct password is passed, the necessary security procedures to make the data accessible are run.

For optional features that you can enable at initialization time, see **Security**, **Multiple User Support**, and **MobileFirst Adapter Integration** in the second part of this module



```

1 Context context = getContext();
2 try {
3     JSONStoreCollection people = new JSONStoreCollection("people");
4     people.setSearchField("name", SearchFieldType.STRING);
5     people.setSearchField("age", SearchFieldType.INTEGER);
6     List<JSONStoreCollection> collections = new LinkedList<JSONStoreCollection>();
7     collections.add(people);
8     WLJSONStore.getInstance(context).openCollections(collections);
9     // handle success
10 } catch(JSONStoreException e) {
11     // handle failure
12 }

```

Get

Use `getCollectionByName` to create an accessor to the collection. You must call `openCollections` before you call `getCollectionByName`.

```

1 Context context = getContext();
2 try {
3     String collectionName = "people";
4     JSONStoreCollection collection = WLJSONStore.getInstance(context).getCollectionByName(collectionName);
5     // handle success
6 } catch(JSONStoreException e) {
7     // handle failure
8 }

```

The variable `collection` can now be used to perform operations on the `people` collection such as `add`, `find`, and `replace`

Add

Use `addData` to store data as documents inside a collection

```

1 Context context = getContext();
2 try {
3     String collectionName = "people";
4     JSONStoreCollection collection = WLJSONStore.getInstance(context).getCollectionByName(collectionName);
5     //Add options.
6     JSONStoreAddOptions options = new JSONStoreAddOptions();
7     options.setMarkDirty(true);
8     JSONObject data = new JSONObject("{\"age: 23, name: 'yoel'}")
9     collection.addData(data, options);
10    // handle success
11 } catch(JSONStoreException e) {
12    // handle failure
13 }

```

Find

Use `findDocuments` to locate a document inside a collection by using a query. Use `findAllDocuments` to retrieve all the documents inside a collection. Use `findDocumentById` to search by the document unique identifier.

```

1 Context context = getContext();
2 try {
3     String collectionName = "people";
4     JSONStoreQueryPart queryPart = new JSONStoreQueryPart();
5     // fuzzy search LIKE
6     queryPart.addLike("name", name);
7     JSONStoreQueryParts query = new JSONStoreQueryParts();
8     query.addQueryPart(queryPart);
9     JSONStoreFindOptions options = new JSONStoreFindOptions();
10    // returns a maximum of 10 documents, default: returns every document
11    options.setLimit(10);
12    JSONStoreCollection collection = WLJSONStore.getInstance(context).getCollectionByName(collectionName);
13    List<JSONObject> results = collection.findDocuments(query, options);
14    // handle success
15 } catch(JSONStoreException e) {
16    // handle failure
17 }

```

Replace

Use `replaceDocument` to modify documents inside a collection. The field that you use to perform the replacement is `_id`, the document unique identifier.

```

1 Context context = getContext();
2 try {
3     String collectionName = "people";
4     JSONStoreCollection collection = WLJSONStore.getInstance(context).getCollectionByName(collectionName);
5     JSONStoreReplaceOptions options = new JSONStoreReplaceOptions();
6     // mark data as dirty
7     options.setMarkDirty(true);
8     JSONStore replacement = new JSONObject("{\"_id: 1, json: {age: 23, name: 'chevy'}}");
9     collection.replaceDocument(replacement, options);
10    // handle success
11 } catch(JSONStoreException e) {
12    // handle failure
13 }

```

This examples assumes that the document `{_id: 1, json: {name: 'yoel', age: 23}}` is in the collection

Remove

Use `removeDocumentById` to delete a document from a collection.

Documents are not erased from the collection until you call `markDocumentClean`. For more information, see the **MobileFirst Adapter Integration** section later in this tutorial

```

1 Context context = getContext();
2 try {
3     String collectionName = "people";
4     JSONStoreCollection collection = WLJSONStore.getInstance(context).getCollectionByName(collectionName);
5     JSONStoreRemoveOptions options = new JSONStoreRemoveOptions();
6     // Mark data as dirty
7     options.setMarkDirty(true);
8     collection.removeDocumentById(1, options);
9     // handle success
10 } catch(JSONStoreException e) {
11     // handle failure
12 }

```

Remove Collection

Use `removeCollection` to delete all the documents that are stored inside a collection. This operation is similar to dropping a table in database terms

```

1 Context context = getContext();
2 try {
3     String collectionName = "people";
4     JSONStoreCollection collection = WLJSONStore.getInstance(context).getCollectionByName(collectionName);
5     collection.removeCollection();
6     // handle success
7 } catch(JSONStoreException e) {
8     // handle failure
9 }

```

Destroy

Use `destroy` to remove the following data:

- All documents
- All collections
- All Stores "See **Multiple User Support** later in this tutorial"
- All JSONStore metadata and security artifacts "See **Security** later in this tutorial"

```

1 Context context = getContext();
2 try {
3     WLJSONStore.getInstance(context).destroy();
4     // handle success
5 } catch(JSONStoreException e) {
6     // handle failure
7 }

```

Advanced Usage

Security

You can secure all the collections in a store by passing a `JSONStoreInitOptions` object with a password to the `openCollections` function. If no password is passed, the documents of all the collections in the store are not encrypted.

Some security metadata is stored in shared preferences (Android);

The store is encrypted with a 256-bit Advanced Encryption Standard (AES) key. All keys are strengthened with Password-Based Key Derivation Function 2 (PBKDF2).

Use `closeAll` to lock access to all the collections until you call `openCollections` again. If you think of `openCollections` as a login function you can think of `closeAll` as the corresponding logout function.

Use `changePassword` to change the password.

```
1 Context context = getContext();
2 try {
3     JSONStoreCollection people = new JSONStoreCollection("people");
4     people.setSearchField("name", SearchFieldType.STRING);
5     people.setSearchField("age", SearchFieldType.INTEGER);
6     List<JSONStoreCollection> collections = new LinkedList<JSONStoreCollection>();
7     collections.add(people);
8     JSONStoreInitOptions options = new JSONStoreInitOptions();
9     options.setPassword("123");
10    WLJSONStore.getInstance(context).openCollections(collections, options);
11    // handle success
12 } catch(JSONStoreException e) {
13    // handle failure
14 }
```

Multiple User Support

You can create multiple stores that contain different collections in a single MobileFirst application. The `openCollections` function can take an options object with a username. If no username is given, the default username is **jsonstore**

```

1 Context context = getContext();
2 try {
3     JSONStoreCollection people = new JSONStoreCollection("people");
4     people.setSearchField("name", SearchFieldType.STRING);
5     people.setSearchField("age", SearchFieldType.INTEGER);
6     List<JSONStoreCollection> collections = new LinkedList<JSONStoreCollection>();
7     collections.add(people);
8     JSONStoreInitOptions options = new JSONStoreInitOptions();
9     options.setUsername("yoel");
10    WLJSONStore.getInstance(context).openCollections(collections, options);
11    // handle success
12 } catch(JSONStoreException e) {
13    // handle failure
14 }

```

MobileFirst Adapter Integration

This section assumes that you are familiar with MobileFirst adapters. MobileFirst Adapter Integration is optional and provides ways to send data from a collection to an adapter and get data from an adapter into a collection.

You can achieve these goals by using functions such as `WLClient.invokeProcedure` or your own instance of an `HttpClient` if you need more flexibility.

Adapter Implementation

Create a MobileFirst adapter and name it **'People'**. Define its procedures `addPerson`, `getPeople`, `pushPeople`, `removePerson`, and `replacePerson`.

```

1  function getPeople() {
2      var data = { peopleList : [{name: 'chevy', age: 23}, {name: 'yoel', age: 23}] };
3      WL.Logger.debug('Adapter: people, procedure: getPeople called.');
```

4 WL.Logger.debug('Sending data: ' + JSON.stringify(data));

```

5      return data;
6  }
7  function pushPeople(data) {
8      WL.Logger.debug('Adapter: people, procedure: pushPeople called.');
```

9 WL.Logger.debug('Got data from JSONStore to ADD: ' + data);

```

10     return;
11 }
12 function addPerson(data) {
13     WL.Logger.debug('Adapter: people, procedure: addPerson called.');
```

14 WL.Logger.debug('Got data from JSONStore to ADD: ' + data);

```

15     return;
16 }
17 function removePerson(data) {
18     WL.Logger.debug('Adapter: people, procedure: removePerson called.');
```

19 WL.Logger.debug('Got data from JSONStore to REMOVE: ' + data);

```

20     return;
21 }
22 function replacePerson(data) {
23     WL.Logger.debug('Adapter: people, procedure: replacePerson called.');
```

24 WL.Logger.debug('Got data from JSONStore to REPLACE: ' + data);

```

25     return;
26 }
```

Load data from MobileFirst Adapter

To load data from a MobileFirst Adapter use `WLClient.invokeProcedure`.

```

1  WLResponseListener responseListener = new WLResponseListener() {
2      @Override
3      public void onFailure(final WLFailResponse response) {
4          // handle failure
5      }
6
7      @Override
8      public void onSuccess(WLResponse response) {
9          try {
10             JSONArray loadedDocuments = response.getResponseJSON().getJSONArray("peopleList");
11             catch(Exception e) {
12                 // error decoding JSON data
13             }
14         }
15     };
16
17     WLProcedureInvocationData invocationData = new WLProcedureInvocationData("People", "getPeople");
18     Context context = getContext();
19     WLClient client = WLClient.createInstance(context);
20     client.invokeProcedure(invocationData, responseListener);
```

Get Push Required (Dirty Documents)

Calling `findAllDirtyDocuments` returns an array of so called "dirty documents", which are documents that have local modifications that do not exist on the back-end system.

```
1 Context context = getContext();
2 try {
3     String collectionName = "people";
4     JSONStoreCollection collection = WLJSONStore.getInstance(context).getCollectionByName(collectionName);
5     List<JSONObject> dirtyDocs = collection.findAllDirtyDocuments();
6     // handle success
7 } catch(JSONStoreException e) {
8     // handle failure
9 }
```

To prevent JSONStore from marking the documents as "dirty", pass the option `options.setMarkDirty(false)` to add, replace, and remove

Push changes

To push changes to a MobileFirst adapter, call the `findAllDirtyDocuments` to get a list of documents with modifications and then use `WLClient.invokeProcedure`. After the data is sent and a successful response is received make sure you call `markDocumentsClean`.

```
1 WLResponseListener responseListener = new WLResponseListener() {
2     @Override
3     public void onFailure(final WLFailResponse response) {
4         // handle failure
5     }
6
7     @Override
8     public void onSuccess(WLResponse response) {
9         // handle success
10    }
11 };
12
13 Context context = getContext();
14 WLClient client = WLClient.createInstance(context);
15 try {
16     String collectionName = "people";
17     JSONStoreCollection collection = WLJSONStore.getInstance(context).getCollectionByName(collectionName);
18     List<JSONObject> dirtyDocuments = collection.findAllDirtyDocuments();
19     WLProcedureInvocationData invocationData = new WLProcedureInvocationData("People", "pushPeople");
20     invocationData.setParameters(new Object[]{dirtyDocuments});
21     client.invokeProcedure(invocationData, responseListener);
22 } catch(JSONStoreException e) {
23     // handle failure
24 }
```




Sample application

Click to download (<https://github.com/MobileFirst-Platform-Developer-Center/JSONStore>) the MobileFirst project.

Click to download (<https://github.com/MobileFirst-Platform-Developer-Center/JSONStoreAndroid>) the Native project.

The Native Android project contains an application that demonstrates the use of JSONStore.

Additional information

For more information about JSONStore, see the product user documentation.