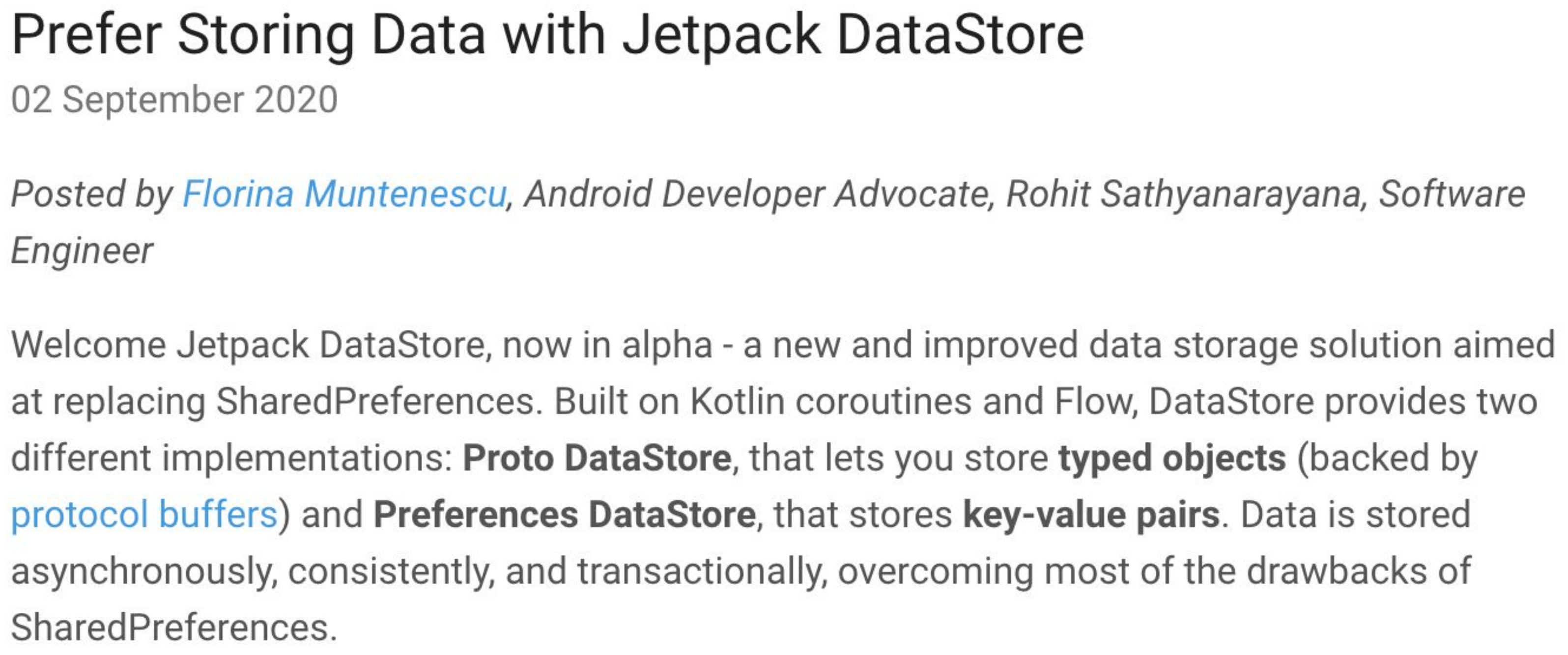
Good Bye SharedPreferences!

By Hendra Wijaya Djiono

Someday on Android Developers Blog



Prefer Storing Data with Jetpack DataStore

02 September 2020

Posted by Florina Muntenescu, Android Developer Advocate, Rohit Sathyanarayana, Software Engineer

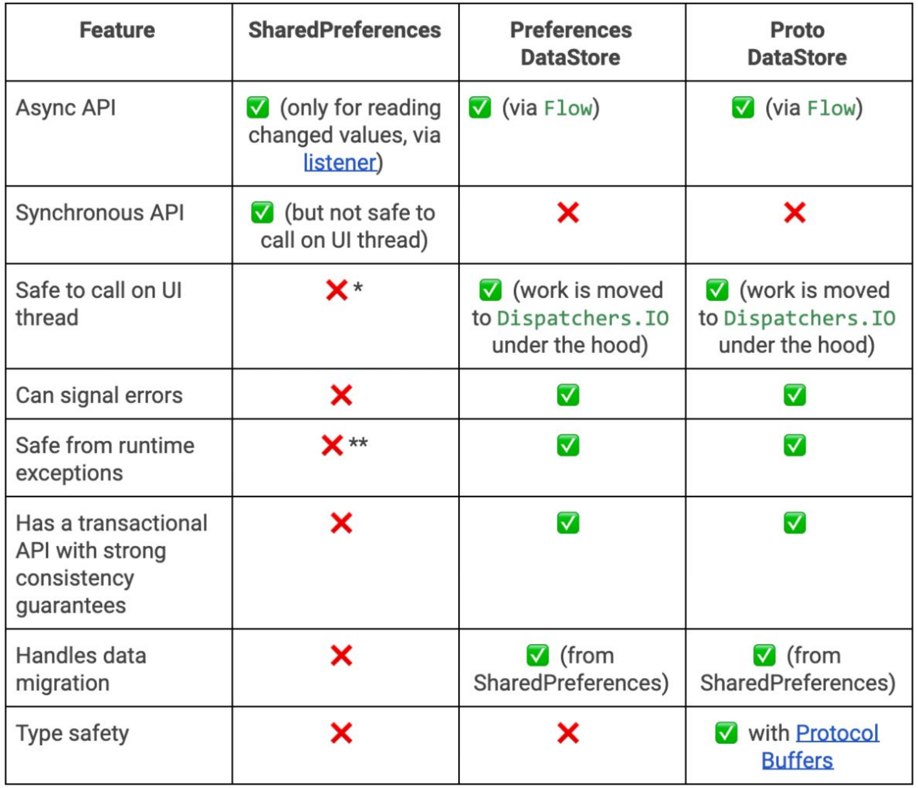
Welcome Jetpack DataStore, now in alpha - a new and improved data storage solution aimed at replacing SharedPreferences. Built on Kotlin coroutines and Flow, DataStore provides two different implementations: **Proto DataStore**, that lets you store **typed objects** (backed by protocol buffers) and **Preferences DataStore**, that stores **key-value pairs**. Data is stored asynchronously, consistently, and transactionally, overcoming most of the drawbacks of SharedPreferences.

DataStore Types

Preferences DataStore, like SharedPreferences, accesses data based on keys, without defining a schema upfront.

Proto DataStore defines the schema using Protocol buffers. Using Protobufs allows **persisting strongly typed data**. They are faster, smaller, simpler, and less ambiguous than XML and other similar data formats. While Proto DataStore requires you to learn a new serialization mechanism, we believe that the strongly typed advantage brought by Proto DataStore is worth it.

SharedPreferences VS DataStore



Feature	SharedPreferences	Preferences DataStore	Proto DataStore
Async API	✅ (only for reading changed values, via listener)	✅ (via `Flow`)	✅ (via `Flow`)
Synchronous API	✅ (but not safe to call on UI thread)	❌	❌
Safe to call on UI thread	❌ *	✅ (work is moved to `Dispatchers.IO` under the hood)	✅ (work is moved to `Dispatchers.IO` under the hood)
Can signal errors	❌	✅	✅
Safe from runtime exceptions	❌ **	✅	✅
Has a transactional API with strong consistency guarantees	❌	✅	✅
Handles data migration	❌	✅ (from SharedPreferences)	✅ (from SharedPreferences)
Type safety	❌	❌	✅ with Protocol Buffers

Should We Replace Room With DataStore?

Depends!

If you have a need for **partial updates**, **referential integrity**, or **large/complex datasets**, you should consider using Room instead of DataStore. DataStore is ideal for small or simple datasets and does not support partial updates or referential integrity.

Preferences DataStore VS SharedPreferences

Preferences DataStore API is similar to **SharedPreferences** with several notable differences:

1. Handles data updates transactionally
2. Exposes a Flow representing the current state of data
3. Does not have data persistent methods (apply(), commit())
4. Does not return mutable references to its internal state
5. Exposes an API similar to Map and MutableMap with typed keys

How Ready is DataStore?

Based on https://developer.android.com/jetpack/androidx/releases/datastore, the latest version is:

Version 1.0.0-rc01, released on June 30, 2021

How stable? Has bugs?

https://issuetracker.google.com/issues?q=componentid:907884%20status:open

Flipper doesn't support datastore yet

https://github.com/facebook/flipper/issues/2407

How To Use DataStore

Import Dependency(ies)

```
// Typed DataStore (Typed API surface, such as Proto)
    dependencies {
        implementation "androidx.datastore:datastore:1.0.0-rc01"

        // optional - RxJava2 support
        implementation "androidx.datastore:datastore-rxjava2:1.0.0-rc01"

        // optional - RxJava3 support
        implementation "androidx.datastore:datastore-rxjava3:1.0.0-rc01"
    }

    // Alternatively - use the following artifact without an Android dependency.
    dependencies {
        implementation "androidx.datastore:datastore-core:1.0.0-rc01"
    }
```

Import Dependency(ies)

```
// Preferences DataStore (SharedPreferences like APIs)
    dependencies {
        implementation "androidx.datastore:datastore-preferences:1.0.0-rc01"

        // optional - RxJava2 support
        implementation "androidx.datastore:datastore-preferences-rxjava2:1.0.0-rc01"

        // optional - RxJava3 support
        implementation "androidx.datastore:datastore-preferences-rxjava3:1.0.0-rc01"
    }

    // Alternatively - use the following artifact without an Android dependency.
    dependencies {
        implementation "androidx.datastore:datastore-preferences-core:1.0.0-rc01"
    }
```

Creating Preferences DataStore

```
private const val PREFERENCES_NAME = "some_preferences"

// “by preferencesDataStore” will return the same object (singleton), so we could
access it from anywhere that have access to “Context”.
val Context.dataStore by preferencesDataStore(
    name = PREFERENCES_NAME
)
```

Migrating from SharedPreferences

```
private const val PREFERENCES_NAME = "some_preferences"

private val Context.dataStore by preferencesDataStore(
    name = PREFERENCES_NAME,
    produceMigrations = { context ->
        // Since we're migrating from SharedPreferences, add a migration based on the
        // SharedPreferences name
        listOf(SharedPreferencesMigration(context, PREFERENCES_NAME))
    }
)
```

Saving Data

```
dataStore.edit { preferences ->
   preferences[stringPreferencesKey("text")] = text
}
```

Note: “edit” is a suspend function.

Supported Key Types

`stringPreferencesKey`

`intPreferencesKey`

`doublePreferencesKey`

`booleanPreferencesKey`

`floatPreferencesKey`

`longPreferencesKey`

`stringSetPreferencesKey`

Loading Data

```
dataStore.data.map { preferences ->
   preferences[/*replace with preference key!*/] ?: "No saved value!"
}
.asLiveData()
       .observe(lifeCycleOwner) { returnedValue ->
           // do something with returnedValue
       }
```

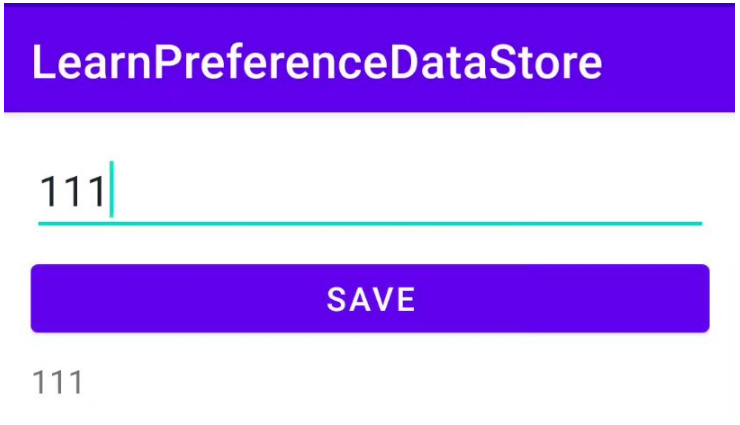
Exception Handling

```
dataStore.data.map { preferences ->
   preferences[/*replace with preference key!*/] ?: "No saved value!"
}.catch { exception ->
   // dataStore.data throws an IOException when an error is encountered when reading data
   if (exception is IOException) {
       emit(emptyPreferences())
   } else {
       throw exception
   }
}
```

Let’s Code!

Build a simple app using **Preferences DataStore** that have an **EditText** to put data, a **Button** to save the data from EditText to **Preferences DataStore**, a **TextView** that will show the data from **Preferences DataStore**.

Sample layout:



Recap

1. SharedPreferences comes with a series of drawbacks - from synchronous API that can appear safe to call on the UI thread, no mechanism of signaling errors, lack of transactional API and more.
2. DataStore is a replacement for SharedPreferences addressing most of the shortcomings of the API.
3. DataStore has a fully asynchronous API using Kotlin coroutines and Flow, handles data migration, guarantees data consistency and handles data corruption.

References

- Prefer Storing Data with Jetpack DataStore https://android-developers.googleblog.com/2020/09/prefer-storing-data-with-jetpack.html
- For more comprehensive codelab https://developer.android.com/codelabs/android-preferences-datastore#0