Data Store API

Link: http://airpingu.github.io/data-store-api/index.html

Author: Gene Lian (clian@mozilla.com)

Motivation

- We are constantly having to revise **Contacts Manager API** and **Messaging API** because their querying capabilities aren't matching what applications need.

WebIDL

```
interface MessagingManager {
    readonly    attribute SmsManager sms;
    readonly    attribute MmsManager mms;
    Promise findMessages (MessagingFilter filter, FilterOptions options);
    Promise findConversations (DOMString groupBy, MessagingFilter filter, FilterOptions options);
```

WebIDL

```
dictionary MessagingFilter {
    MessageType             type;
    Date                    startDate;
    Date                    endDate;
    DOMString               from;
    sequence<DOMString>     recipients;
    (SmsState or MmsState)  state;
    DOMString               serviceID;
    boolean                 read;
};
```

WebIDL

```
dictionary FilterOptions {
    DOMString      sortBy;
    DOMString      sortOrder;
    unsigned long  limit;
};
```

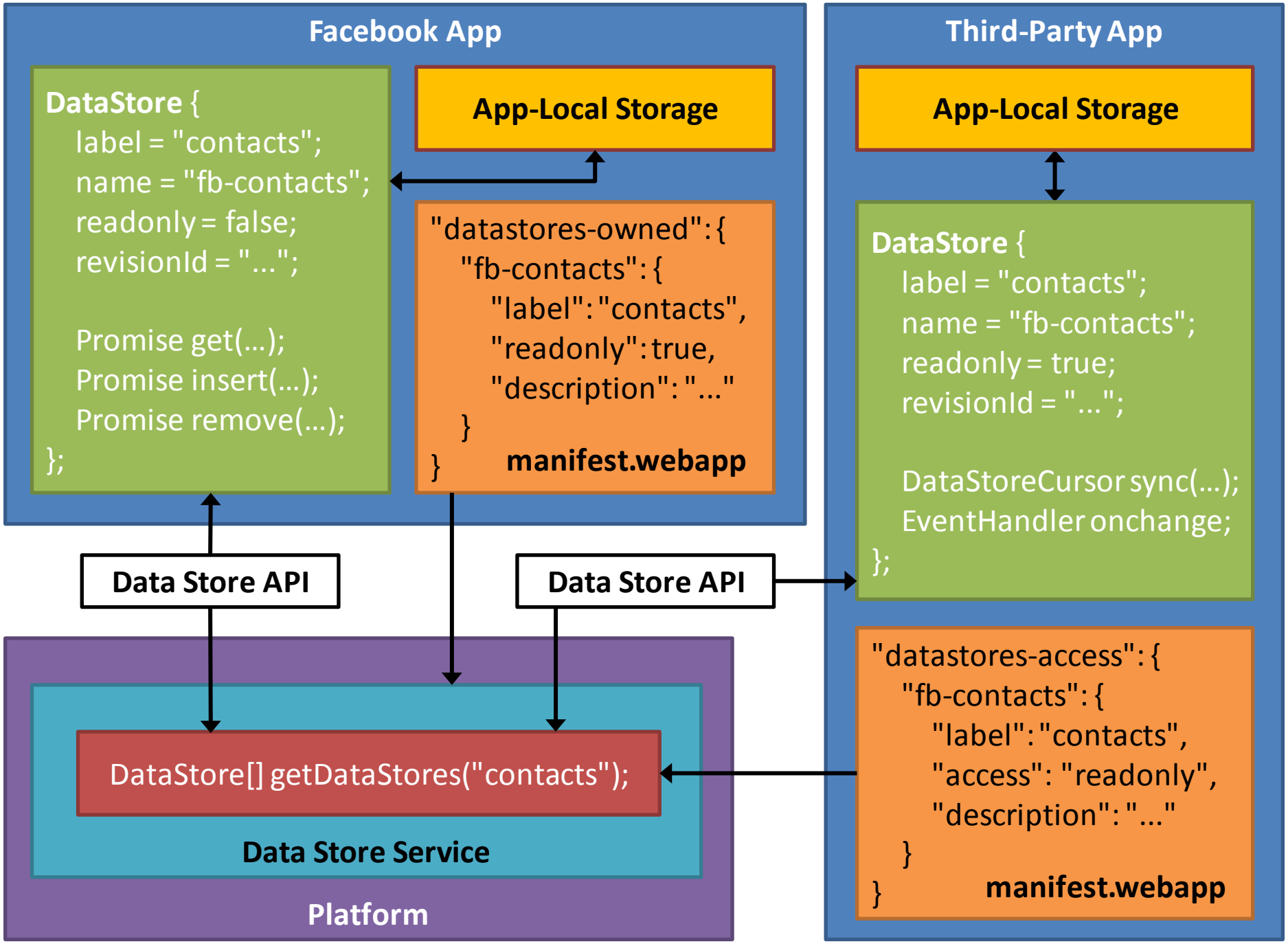
It's an endless task to define how to filter for various apps!

Solution

- **Data Store API** to allow apps to **synchronize** data from a central data store into their local storages which can be in various formats for filtering.

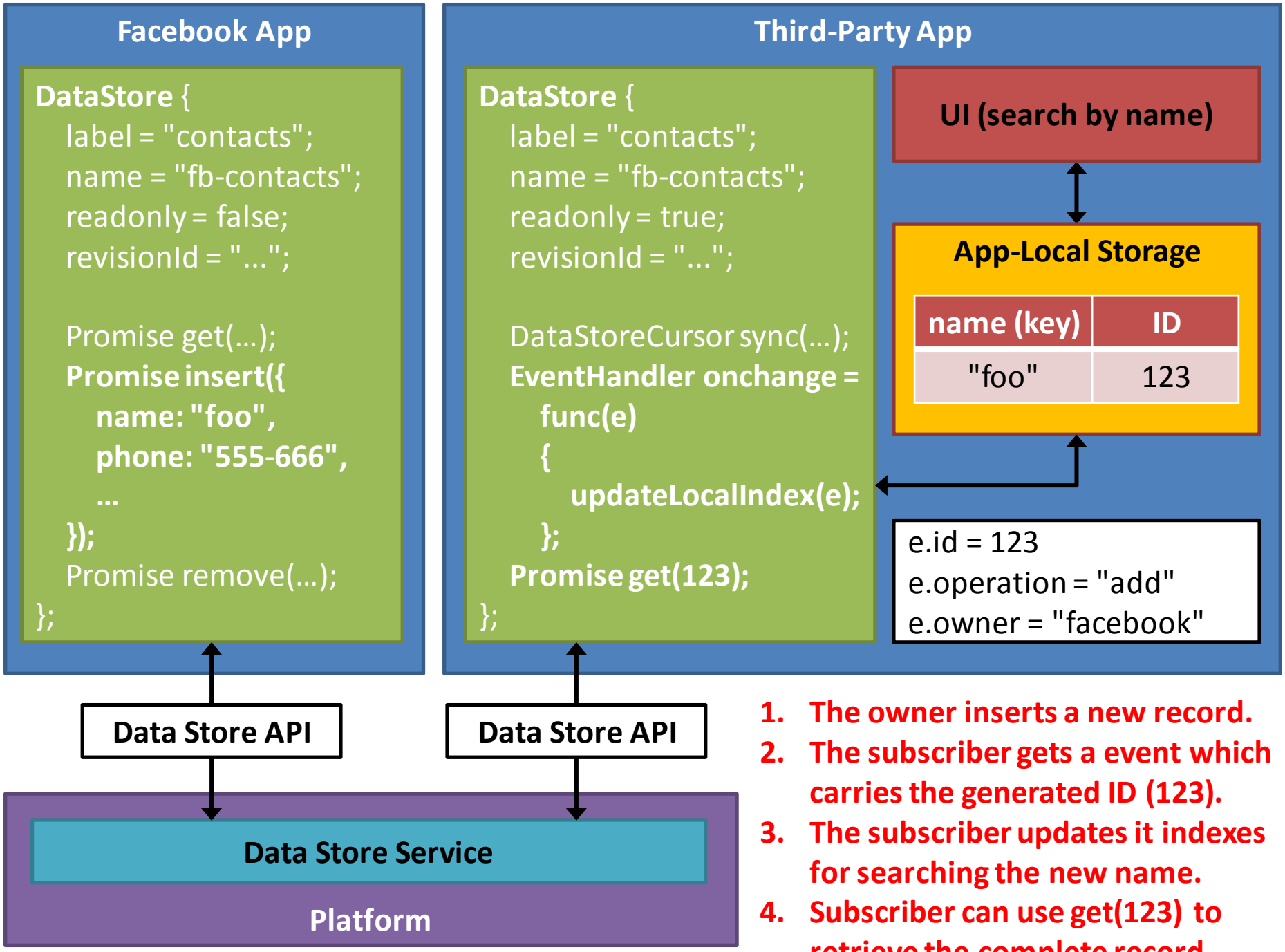
WebIDL

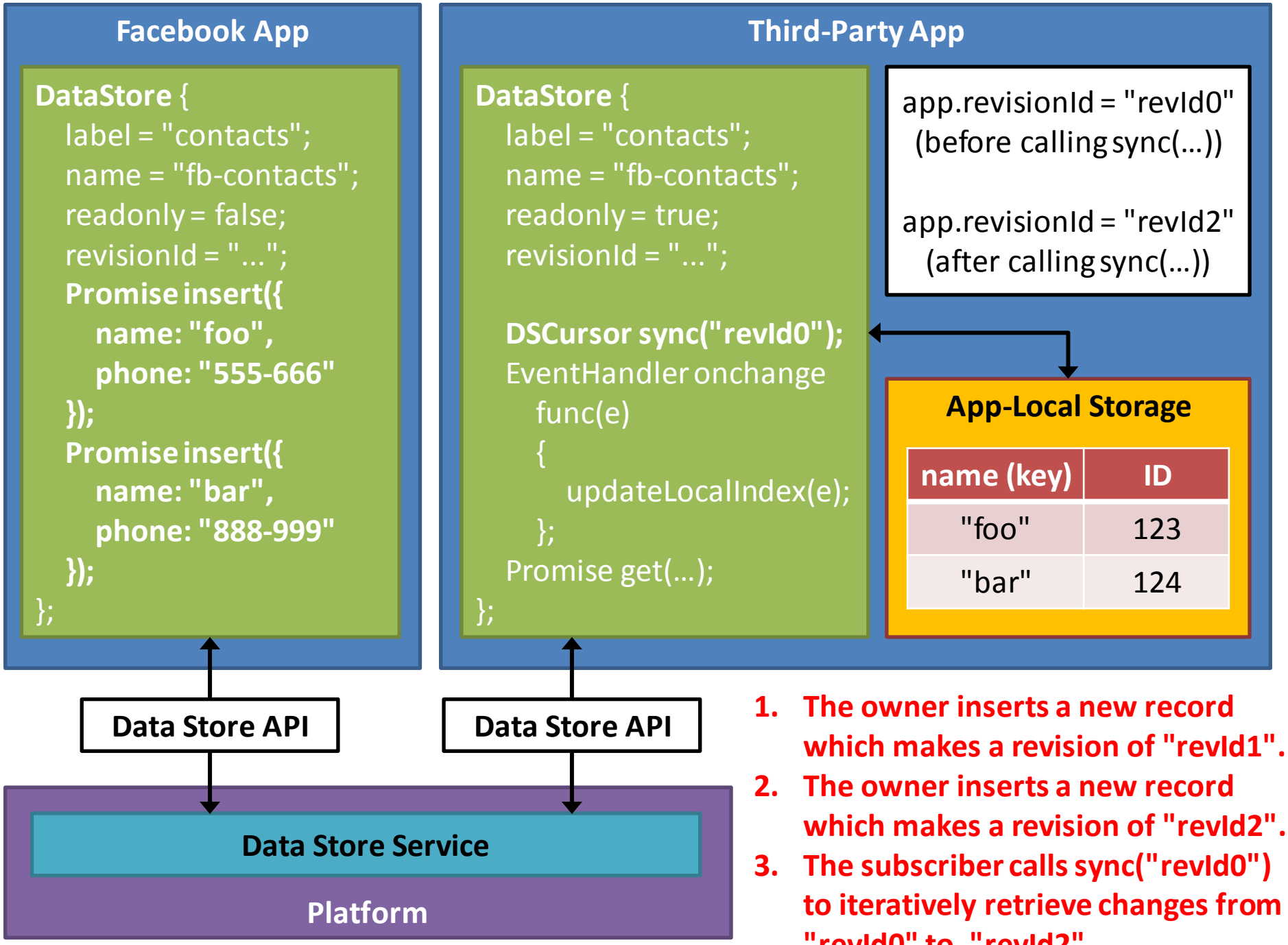
```
interface DataStore : EventTarget {
    readonly    attribute DOMString    label;
    readonly    attribute DOMString    name;
    readonly    attribute DOMString    owner;
    readonly    attribute boolean      readOnly;
    readonly    attribute DOMString    revisionId;
    Promise         get (DataStoreKey... id);
    Promise         update (any data, DataStoreKey id);
    Promise         insert (any data);
    Promise         remove (DataStoreKey id);
    Promise         clear ();
    Promise         getLength ();
    DataStoreCursor sync (optional DOMString revisionId);
                    attribute EventHandler onchange;
};
```


Facebook App
DataStore {
label = "contacts";
name = "fb-contacts";
readonly = false;
revisionId = "...";
Promise get(...);
Promise insert(...);
Promise remove(...);
};
App-Local Storage
"datastores-owned":{
"fb-contacts":{
"label":"contacts",
"readonly":true,
"description":"..."
}
}
manifest.webapp
Third-Party App
App-Local Storage
DataStore {
label = "contacts";
name = "fb-contacts";
readonly = true;
revisionId = "...";
DataStoreCursor sync(...);
EventHandler onchange;
};
Data Store API
Data Store API
DataStore[] getDataStores("contacts");
Data Store Service
Platform
"datastores-access":{
"fb-contacts":{
"label":"contacts",
"access": "readonly",
"description":"..."
}
}
manifest.webapp

How to synchronize?

- **Data Store API** provides basic methods to add, retrieve and delete data but it doesn't expose methods for building indexes and filtering data.
- Instead, it provides a **synchronizing mechanism** for apps to keep their local storages up-to-date and accordingly update the local indexes needed for filtering data in their own ways.
  - **DataStoreCursor sync** (**DOMString** *revisionId*);
  - **EventHandler onchange**;


Facebook App
DataStore {
label = "contacts";
name = "fb-contacts";
readonly = false;
revisionId = "...";
Promise get(...);
Promise insert({
name: "foo",
phone: "555-666",
...
});
Promise remove(...);
};
Third-Party App
DataStore {
label = "contacts";
name = "fb-contacts";
readonly = true;
revisionId = "...";
DataStoreCursor sync(...);
EventHandler onchange =
func(e)
{
updateLocalIndex(e);
};
Promise get(123);
};
UI (search by name)
App-Local Storage
name (key)
ID
"foo"
123
e.id = 123
e.operation = "add"
e.owner = "facebook"
Data Store API
Data Store API
Data Store Service
Platform
1. The owner inserts a new record.
2. The subscriber gets a event which carries the generated ID (123).
3. The subscriber updates it indexes for searching the new name.
4. Subscriber can use get(123) to retrieve the complete record.


Facebook App
DataStore {
label = "contacts";
name = "fb-contacts";
readonly = false;
revisionId = "...";
Promise insert({
name: "foo",
phone: "555-666"
});
Promise insert({
name: "bar",
phone: "888-999"
});
};
Third-Party App
DataStore {
label = "contacts";
name = "fb-contacts";
readonly = true;
revisionId = "...";
DSCursor sync("revId0");
EventHandler onchange
func(e)
{
updateLocalIndex(e);
};
Promise get(...);
};
app.revisionId = "revId0"
(before calling sync(...))
app.revisionId = "revId2"
(after calling sync(...))
App-Local Storage
name (key)
ID
"foo"
123
"bar"
124
Data Store API
Data Store API
Data Store Service
Platform
1. The owner inserts a new record which makes a revision of "revId1".
2. The owner inserts a new record which makes a revision of "revId2".
3. The subscriber calls sync("revId0") to iteratively retrieve changes from "revId0" to "revId2".

Issues (1/2)

- How it works for Messaging API?
  - System is the owner of the built-in message data stores and messaging apps are the consumers which synchronize data into local indexes.

```
interface DataStore {
    label = "messaging";
    name = "messaging-sim-1";
    owner = "sim-1";
}
```

```
interface DataStore {
    label = "messaging";
    name = "messaging-sim-2";
    owner = "sim-2";
}
```

Issues (2/2)

- How it works for Contacts Manager API?
  - System is the owner of the built-in contacts data stores and contacts apps are the consumers which synchronize data into local indexes.

```
interface DataStore {
    label = "contacts";
    name = "contacts-sim-1";
    owner = "sim-1";
}
```

```
interface DataStore {
    label = "contacts";
    name = " contacts-sim-2";
    owner = "sim-2";
}
```

Q&A

- Thank you!