

LocalStorage array handling



how to save and retrieve an array of object in local storage using electron js

In an Electron.js app, saving and retrieving an **array of objects** in **localStorage** is similar to how you'd do it in a regular web app using the `window.localStorage` API — but remember, **localStorage is only available in the renderer process** (i.e., inside the web pages or frontend).

Here's a simple step-by-step guide to **save and retrieve an array of objects** using `localStorage` in Electron's renderer process:

✓ Saving an Array of Objects

```
javascript

const dataArray = [
  { id: 1, name: "Item 1", quantity: 5 },
  { id: 2, name: "Item 2", quantity: 10 }
];

// Convert array to JSON string and save
localStorage.setItem("myDataArray", JSON.stringify(dataArray));
```

✓ Retrieving the Array of Objects

```
javascript

const storedData = localStorage.getItem("myDataArray");

if (storedData) {
  const parsedArray = JSON.parse(storedData);
  console.log(parsedArray);
  // Use the array as needed
} else {
  console.log("No data found");
}
```

✓ Notes

- **localStorage only stores strings**, so you must use `JSON.stringify()` when saving and `JSON.parse()` when retrieving.
- This data is persistent between app restarts **as long as the renderer session is not cleared**.

! If You Need to Use from Main Process (Alternative)

If you're in the main process and need to persist data, you **cannot use localStorage directly**. Instead, consider using:

- [electron-store](#)
- `fs` (filesystem) for manual JSON file storage

Example using **electron-store** in main process:

```
bash
```

```
npm install electron-store
```

```
js
```

```
// main.js or preload.js
const Store = require('electron-store');
const store = new Store();

const myArray = [
  { id: 1, name: "Item A" },
  { id: 2, name: "Item B" }
];

store.set('items', myArray);


// Later to retrieve:
const savedItems = store.get('items');
console.log(savedItems);
```

Let me know if you're working in **main** or **renderer** and want help setting up a storage method accordingly.



what is the limit of data size of above kind in the local storage

The **data size limit** for `localStorage` in Electron.js (and in general browser environments) is usually around:

 **5 MB per origin (domain + protocol)**

Details:

- **Unit:** That's **~5 megabytes** of character data (since `localStorage` stores everything as strings).
- **Scope:** The limit applies **per origin**, so it's shared among all keys under the same domain/protocol combination (e.g., `file://` in Electron).
- **Encoding:** Because `localStorage` uses UTF-16 encoding (2 bytes per character), your actual number of characters will be **about 2.5 million** characters (assuming simple ASCII text).

Important Caveats in Electron:

- In Electron apps using `file://` protocol (default for local HTML), the **origin** is considered `file://`, so all `localStorage` data for the entire app shares the same 5 MB limit.
- If you exceed this limit, `localStorage.setItem()` will silently fail or throw a `QuotaExceededError`.

🔴 Downsides of Using `localStorage` for Large Data:

- **No structure:** Only strings — you must serialize/deserialize manually.
- **No async:** It's synchronous and blocks the main thread.
- **Small capacity:** Limited to ~5MB.

✅ Alternative for Larger or Structured Data:

Use `electron-store` or `lowdb` or directly write to a JSON/SQLite file.

Example: `electron-store`

```
bash
```

```
npm install electron-store
```

```
js
```

```
const Store = require('electron-store');
const store = new Store();
```

```
store.set('largeData', yourLargeArray); // no size limit unless disk is full
```

This has no 5MB limit and is more robust.

Let me know if you need a complete example using `electron-store` or want to handle large data chunks in a specific format like images, blobs, etc.



is `electron-store` is newly added to electron

No, `electron-store` is **not part of Electron itself**, and it is **not newly added** to Electron.

✅ Summary:

- `electron-store` is a **third-party Node.js module**, not an official or built-in part of Electron.
- It has been around **since 2016**, maintained by the open-source community.
- You must **install it manually** using `npm`:

```
bash
```

```
npm install electron-store
```

What electron-store Does:

- Provides **easy persistent key-value storage**.
- Stores data in a **JSON file** (usually in `appData` directory like `~/.config/<app-name>` or `%APPDATA%`).
- Works in **main** and **preload** processes (not directly in the renderer unless exposed via IPC).
- Automatically handles **JSON serialization**, **type safety**, and **data migration**.

Example Use (main or preload process):

```
js

const Store = require('electron-store');
const store = new Store();

// Save data
store.set('user', { name: "Kaushiq", role: "admin" });

// Read data
const user = store.get('user');
console.log(user.name); // "Kaushiq"
```

Why Use It?

- No 5MB limit like `localStorage`
- Works across restarts
- Easy to read/write structured data
- Cross-platform (Windows, macOS, Linux)

Let me know if you want a complete setup with `electron-store` using preload + renderer (IPC) for best practice.



what do you mean by unless disk is full. do you mean hard disk

Yes — when I said "**unless disk is full**," I meant:

 **Your computer's hard disk or SSD (storage drive) is full.**

Why It Matters:

When using `electron-store`, data is saved to a **local JSON file** on disk. So:

- You can store **as much data as your disk space allows**.
- Unlike `localStorage` (which has a ~5MB limit), `electron-store` doesn't impose a strict size cap.
- The **only practical limit** is how much free space is available on your hard drive.

🧠 Example:

If your app stores 500MB of cached objects using **electron-store**, that's fine — as long as the disk has enough free space. But:

- If your disk runs out of space (e.g., 0 bytes free), **electron-store** **can no longer save data**.
- In such cases, write attempts may **fail silently** or throw an error depending on the OS.

🔧 Optional: Error Handling

You can catch write errors like this:

```
js
try {
  store.set('myBigData', largeObject);
} catch (error) {
  console.error('Failed to save data:', error);
}
```

Let me know if you want help checking disk space before saving large data!



how to call store function from renderer process

To call a function like `store.set()` from the **renderer process** in Electron, you must use **IPC (Inter-Process Communication)** — because:

! **electron-store** should be used in the **main** or **preload** process, not directly in the renderer process (for security and architectural reasons).

✅ Safe Way to Access electron-store from Renderer:

🔧 Step 1: Setup in preload.js

Expose limited, safe API to the renderer:

```
js
// preload.js
const { contextBridge, ipcRenderer } = require('electron');

contextBridge.exposeInMainWorld('electronAPI', {
  setStore: (key, value) => ipcRenderer.send('store-set', { key, value }),
  getStore: (key) => ipcRenderer.invoke('store-get', key)
});
```

Make sure `preload.js` is properly linked in `main.js` when creating your `BrowserWindow`:

```
js

// main.js
mainWindow = new BrowserWindow({
  webPreferences: {
    preload: path.join(__dirname, 'preload.js'),
    contextIsolation: true,
    nodeIntegration: false,
  }
});
```

Step 2: Handle Store in main.js

Use `electron-store` here:

```
js

// main.js
const { ipcMain } = require('electron');
const Store = require('electron-store');
const store = new Store();

ipcMain.on('store-set', (event, { key, value }) => {
  store.set(key, value);
});

ipcMain.handle('store-get', (event, key) => {
  return store.get(key);
});
```

Step 3: Use in Renderer (e.g., in your HTML/JS)

```
html

<script>
  // Set data
  window.electronAPI.setStore('user', { name: 'Kaushiq', loggedIn: true });

  // Get data
  window.electronAPI.getStore('user').then((user) => {
    console.log('User from store:', user);
  });
</script>
```

Why Use This Pattern?

- Keeps your app **secure** by not exposing full Node.js in the renderer.
- Uses `contextBridge` + `ipcRenderer` for **safe, controlled communication**.
- Makes your app more **maintainable and modular**.

Let me know if you want this wrapped in a reusable module or need bi-directional syncing!