RIDNET OS UI dApp Developer Guide

Version 1.0

1. Introduction

Welcome to the GRIDNET OS UI dApp Developer Guide! This document provides essential guidelines, architectural insights, and best practices for creating user interface-driven Decentralized Applications (UI dApps) that run natively within the GRIDNET OS windowing system.

GRIDNET OS offers a unique environment where web technologies (HTML, CSS, JavaScript) are used to build applications that operate within a decentralized framework. Understanding the specific architecture and adhering to the development rules outlined here is crucial for building responsive, secure, and compatible UI dApps.

Target Audience: Web developers (especially those familiar with JavaScript ES6, HTML5, CSS3) aiming to build applications for the GRIDNET OS platform.

Key Goals of this Guide:

- Explain the fundamental architecture of GRIDNET OS UI dApps.
- Detail the **critical CSS and JavaScript rules** required for compatibility and compartmentation.
- Guide developers on how to interact with the core GRIDNET OS functionalities via CVMContext.
- Provide a starting point using the official UI dApp template.
- Outline the deployment process.

2. GRIDNET OS UI dApp Architecture

image810×588 22.9 KB

GRIDNET OS UI dApps operate within a distinct architectural paradigm:

- **Windowed Environment:** Each UI dApp runs inside its own dedicated window, managed by the GRIDNET OS Window Manager. This is distinct from a standard webpage occupying the entire browser viewport.
- **Class-Based Structure:** Every UI dApp is implemented as a JavaScript ECMA6 class that **must** inherit from the CWindow base class provided by the system (/lib/window.js).
- **Shadow DOM Encapsulation:** To ensure strict isolation and prevent conflicts between dApps or with the OS shell, each CWindow instance (and thus each UI dApp instance) renders its content within a dedicated **Shadow DOM** tree. This is a cornerstone of GRIDNET OS UI development.

---

image811×481 26.7 KB

- **CVMContext Singleton:** The CVMContext object (/lib/VMContext.js) is the **single point of entry** for all interactions with the underlying GRIDNET OS. It's a singleton accessed via CVMContext.getInstance(). UI dApps **must not** attempt to bypass CVMContext to interact with system resources.
- **Event-Driven:** Interaction with CVMContext and the OS is heavily event-driven. dApps register listeners for specific events (e.g., new blockchain data, file system updates, network messages) and react accordingly via callback functions.

```
+---------------------------------+      +----------------------------+      +-------------------------+
|       Your UI dApp           |------>|    CVMContext Singleton     |------>|     GRIDNET OS Core      |
| (Instance of CMyDApp          |<------|   (JavaScript API Gateway)  |<------|   (Blockchain, DFS,
|
|  extends CWindow)             |       |   - Provides APIs          |       |    Networking, Threads)  |
|  - Lives in dedicated Shadow DOM|       |   - Manages State          |       |                         |
|  - Interacts ONLY via CVMContext|      |   - Event Dispatch        |      |                    |
+---------------------------------+     +----------------------------+     +--------------------------+
      |                                                   |
      | Renders Into                                      | Communicates Via
      v                                                   v
+---------------------------------+                       +--------------------------+
|      Browser Main Document     |                           | WebSocket / WebRTC      |
|     (GRIDNET OS Shell)        |                           | (Managed by CVMContext)  |
+---------------------------------+                       +--------------------------+
```

3. Getting Started: The UI dApp Template

The fastest way to bootstrap a new UI dApp is to use the official template.

- **Template File:** AppTemplate.js (Located at: GRIDNETOS/WebUI/dApps/AppTemplate.js at main · GRIDNETOS/GRIDNETOS · GitHub)
- **Sample Deployable Package:** UserDeployable.app (Located at: GRIDNETOS/WebUI/dApps/UserDeployable.app at main · GRIDNETOS/GRIDNETOS · GitHub)

Steps to Customize the Template:

1. **Copy & Rename:** Copy AppTemplate.js and rename it (e.g., MyCoolDApp.js). Change the file extension to .app for deployment later.
2. **Rename Class:** Find class CUIAppTemplate extends CWindow and change CUIAppTemplate to your dApp's class name (e.g., class CMyCoolDApp extends CWindow). Replace all other occurrences of CUIAppTemplate within the file.
3. **Set Window Title:** In the constructor, find the super(...) call and change the title parameter (e.g., super(positionX, positionY, width, height, myCoolDAppBody, "My Cool dApp", CMyCoolDApp.getIcon(), true);).
4. **Update Export:** Change the last line from export default CUIAppTemplate; to export default CMyCoolDApp; (using your class name).
5. **Define Icon:** Implement the static getIcon() method. Return a Base64 encoded data:image/png;base64,... string for your dApp's icon.

6. **Define HTML Body:** Modify the apptemplateBody variable to hold your dApp's primary HTML structure. Alternatively, load HTML content from an external file (ensure relative paths are correct for bundling/deployment). Remember to include standard stylesheets if needed within this HTML: .
7. **Set Package ID:** Implement the static getPackageID() method. Return a unique, reverse-domain-style identifier (e.g., return "com.mycompany.mycooldapp";). **Must** start with org.gridnetproject.UIdApps. for user-deployable apps.
8. **(Optional) Set Category:** Implement static getDefaultCategory() to categorize your app (e.g., 'productivity', 'games', 'utilities'). Default is 'dApp'.
9. **(Optional) Define File Handlers:** Implement static getFileHandlers() to register file types your dApp can open (see CContentHandler in AppTemplate.js).

4. Development Guidelines (CRITICAL)

Adherence to these JavaScript and CSS rules is **mandatory** due to the Shadow DOM encapsulation and multi-instance nature of GRIDNET OS UI dApps. These rules primarily stem from the **Blockchain Explorer UI dApp Implementation Specification (Section 1.2.2 & 1.2.6)** .

4.1 JavaScript Rules

1. **Global Scope Pollution: DO NOT** define variables, functions, or objects in the global (window) scope.

- **Rationale:** Prevents collisions between multiple instances of your dApp or other system components running in the same browser tab.
- **DO:** Encapsulate all state and logic within your dApp's class instance (this). Use modules for organization if needed (but bundle for deployment).

2. **Direct Global DOM Access: DO NOT** use document.getElementById, document.querySelector, document.querySelectorAll.

- **Rationale:** These operate on the main document, outside your dApp's isolated Shadow DOM. They will not find your dApp's elements and could interfere with the OS shell.
- **DO:** Use the CWindow base class methods for scoped DOM access:
  - this.getControl('elementId'): Retrieves an element by its ID within your dApp's Shadow DOM.

    - this.getBody: Returns the root element container within your Shadow DOM where your HTML body resides.
    - this.getBody.querySelector('.my-class') / this.getBody.querySelectorAll('button'): Select elements using CSS selectors, scoped to your dApp's body.

- **Example (Bad):** const btn = document.getElementById('myButton');
- **Example (Good):** const btn = this.getControl('myButton');
- **Example (Good):** const labels = this.getBody.querySelectorAll('.label-class');

3. **Single Module Deployment: DO NOT** rely on multi-file JavaScript deployments.

- **Rationale:** The GRIDNET OS PackageManager expects a single .app file (which is essentially your bundled JS).
- **DO:** Use bundlers (like Rollup, Webpack, Parcel) during your development workflow to combine your class and any non-system-provided dependencies into one file before renaming to .app. You can use import for system-provided modules (/lib/*) and potentially pre-loaded libraries (like Tabulator/Plotly).

4. **Private Data (WeakMap Recommended): DO** use WeakMap to store private instance data.

- **Rationale:** Provides better encapsulation and memory management than closures in some cases, preventing accidental data leaks between instances.
- **Example:** See Blockchain Explorer Specification 1.2.6.8 or the example in the previous response.

5. **Strict Mode: DO** use 'use strict'; at the beginning of your module.

6. **CVMContext Interaction: DO** use CVMContext.getInstance() for **all** interactions with GRIDNET OS features (blockchain, DFS, network, etc.).

- **Rationale:** It's the designated, secure, and managed gateway.
- **DO:** Handle the asynchronous nature of API calls (async/await or Promises).
- **DO:** Register and unregister event listeners correctly (vmContext.add*Listener(this.myCallback.bind(this), this.mID); … vmContext.unregisterEventListenerByID(…)). **Always bind this** for class methods used as callbacks.

7. **System Libraries: DO NOT** bundle libraries provided by GRIDNET OS (e.g., Tabulator.js, Plotly.js mentioned in the spec). Reference them directly via import if needed; the OS makes them available.

4.2 CSS Rules

1. **Viewport Units: DO NOT** use vw or vh units.

- **Rationale:** These are relative to the browser viewport, not your dApp's CWindow container. They will break responsiveness within GRIDNET OS.

- **DO:** Use percentages (%) for layout widths/heights relative to the container. Use em or rem for font sizes, margins, padding, and component spacing.

2. **Window-Relative Positioning: DO NOT** use position: fixed or position: sticky in ways that depend on the main browser window.

- **Rationale:** Your dApp is contained within a CWindow; positioning must respect this container.

- **DO:** Use position: relative, position: absolute scoped within your dApp's elements. Normal document flow (static, default) is preferred.

3. **Fluid Layouts: DO** use Flexbox (display: flex) and/or CSS Grid (display: grid) with relative units (%, fr, em, rem) to create layouts that adapt to the dApp's CWindow dimensions.

4. **Pixel-Perfect Layouts: DO NOT** rely on absolute pixel positioning for major layout elements.

- **Rationale:** Breaks responsiveness.

- **DO:** Use sparingly for fine-tuning element placement inside a component (e.g., an icon within a button).

5. **Avoid !important: DO NOT** use !important excessively.

- **Rationale:** It breaks the CSS cascade and makes styles hard to override or debug. Use more specific selectors if needed.

6. **Scoped Styles:** All your CSS is confined to your dApp's Shadow DOM.

- **DO:** Define all styles within your dApp’s HTML/CSS payload. You don’t need to worry about class name collisions with other dApps.
- **DO NOT:** Rely on external stylesheets (like Bootstrap CDN) unless explicitly loaded inside your Shadow DOM or guaranteed to be provided by the OS environment (unlikely for general styling).
- **DO:** Ensure any third-party UI components you use are compatible with Shadow DOM styling.

7. **CSS Variables: DO NOT** rely on CSS Variables (–my-color: blue;) defined outside your dApp’s Shadow DOM. Define any variables you need within your dApp’s scope.

5. Interacting with GRIDNET OS via CVMContext

- **Access:** const vmContext = CVMContext.getInstance();
- **Asynchronous Calls:** Most data retrieval functions are asynchronous. Use async/await:

```
async initialize() {
  try {
    this.showLoading(true); // Assuming a method to show a loading indicator
    const status = await this.vmContext.getBlockchainStatusA(this.getSystemThreadID(), this,
eVMMetaCodeExecutionMode.RAW);
    this.updateUIWithStatus(status.data); // Assuming .data holds the actual status object
  } catch (error) {
    this.handleError(error);
  } finally {
    this.showLoading(false);
  }
}
```

Event Listeners: Subscribe to updates:

```
// In constructor or initialize
this.myListenerId =
this.vmContext.addVMStateChangedListener(this.onVMStateChange.bind(this), this.mID);
```

```
// Callback method
onVMStateChange(eventData) {
  console.log("VM State Changed:", eventData.state);
  // Update UI based on eventData
}

// In closeWindow()
this.vmContext.unregisterEventListenerByID(this.myListenerId);
```

- **Data Types:** Be aware of the data types returned (JS Objects, ArrayBuffer, BigInt). Use CTools for conversions where necessary.

6. Styling and Responsiveness

- **Target:** Respond fluidly to the CWindow container size. Test by resizing the dApp window in GRIDNET OS.
- **Libraries:** Style components like Tabulator.js or Plotly.js within your dApp's CSS to match the OS theme (e.g., the cyberpunk theme from the Blockchain Explorer spec).
- **Units:** Rely on %, em, rem, flex, grid.

7. Deployment

Deploying your UI dApp involves these steps (as detailed in the Hello World Tutorial):

1. **Bundle:** Ensure all your custom JavaScript code and necessary (non-system) libraries are bundled into a **single JavaScript file** .
2. **Rename:** Rename your bundled JS file to have the .app extension (e.g., MyCoolDApp.app).
3. **Upload:** Connect to the GRIDNET OS DUI (https://ui.gridnet.org or your local instance). Drag and drop your .app file onto the Desktop or into the File Manager UI dApp.
4. **Analyze & Install:** The OS will analyze the package. Events will appear in the bottom log pane. Once analyzed, an icon for your dApp should appear on the Desktop. You can run it in this “sandbox” mode.

5. **(Optional) Commit to Network:** To make the dApp persistent and available across the network, select the .app file in the File Manager and use the ⋮⋮⋮Magic Button to commit it to the Decentralized File System (DFS). This requires GNC for storage fees.

8. Conclusion

Building UI dApps for GRIDNET OS requires understanding its unique windowed, Shadow DOM-based architecture and adhering strictly to the CSS and JavaScript guidelines, primarily for compartmentation and responsiveness. By extending the CWindow class, interacting solely through CVMContext, and following the rules outlined here, you can create powerful, isolated, and responsive decentralized applications. Remember to consult the AppTemplate.js and existing dApp examples (like the Blockchain Explorer) for practical implementation details.

Other Related Resources:

- UI dApp Development Tutorial
- CVMContextDocumentation