Chrome Extension Module Separation Cheat Sheet

🏗️ Basic IIFE Module Pattern

javascript

```
// modules/YourModule.js
(function() {
    'use strict';

    console.log('YourModule: Starting initialization...');

    class YourModule {
        constructor(dependencies) {
            this.deps = dependencies;
            console.log('YourModule: Instance created');
        }

        // Your methods here
        someMethod() {
            return "Hello from module!";
        }
    }

    // Attach to global window object
    window.ePTW_YourModule = YourModule;
    console.log('YourModule: Successfully attached to window');
})();
```

📁 File Structure

text

```
extension/
├── manifest.json
├── content.js          (main orchestrator)
├── modules/
│   ├── AppState.js     (core state management)
│   ├── RoleManager.js  (role operations)
│   ├── OverrideManager.js (SOC operations)
│   └── ... (other modules)
├── assets/
│   └── README_SOC_OVERRIDES.txt
└── popup/
    ├── popup.html
    ├── popup.js
    └── popup.css
```

🔧 Manifest Configuration

json

```
{
    "content_scripts": [{
        "matches": ["*://your-domain.com/*"],
        "js": [
            "modules/AppState.js",
            "modules/RoleManager.js",
            "modules/OverrideManager.js",
            "content.js"
        ],
        "run_at": "document_idle"
    }],
    "web_accessible_resources": [{
        "resources": ["assets/README_SOC_OVERRIDES.txt"],
        "matches": ["*://your-domain.com/*"]
    }]
}
```

🔄 Module Dependencies Pattern

javascript

```
// modules/AppState.js (Base module)
(function() {
    class AppState {
        constructor() {
            this.cache = new Map();
        }

        sharedMethod() {
            return "Shared functionality";
        }
    }

    window.ePTW_AppState = AppState;
})();

// modules/FeatureManager.js (Dependent module)
(function() {
    class FeatureManager {
        constructor(appState) {
            this.appState = appState;  // Dependency injection
        }

        featureMethod() {
            return this.appState.sharedMethod() + " with features";
        }
    }

    window.ePTW_FeatureManager = FeatureManager;
})();
```

🎯 Main Content Script Orchestrator

javascript

```
// content.js
class ContentScriptManager {
    constructor() {
        this.modules = {};
    }

    async initialize() {
        // 1. Check module availability
        this.checkModules();

        // 2. Initialize core module first
        if (window.ePTW_AppState) {
            this.modules.appState = new window.ePTW_AppState();
        }

        // 3. Initialize dependent modules
        if (window.ePTW_FeatureManager && this.modules.appState) {
            this.modules.featureManager = new
window.ePTW_FeatureManager(this.modules.appState);
        }

        // 4. Set up message handling
        chrome.runtime.onMessage.addListener(this.handleMessage.bind(this));
    }

    checkModules() {
        console.log('Available modules:');
        Object.keys(window)
            .filter(key => key.startsWith('ePTW_'))
            .forEach(module => console.log('-', module));
    }

    handleMessage(request, sendResponse) {
        // Route messages to appropriate modules
    }
}
```

⚡ Pre-loading Assets Pattern

javascript

```
// content.js - Asset pre-loading
window.ePTW_ASSETS = {};

async function preloadAssets() {
    try {
        // Pre-load README
        const readmeUrl = chrome.runtime.getURL('assets/README_SOC_OVERRIDES.txt');
        const response = await fetch(readmeUrl);
        window.ePTW_ASSETS.README = await response.text();
        console.log('Assets pre-loaded successfully');
    } catch (error) {
        console.error('Asset pre-loading failed:', error);
    }
}

// Call during initialization
preloadAssets();
```

🚦 Module Loading Order Rules

✅ CORRECT Order:

1. **Base modules** first (AppState, utilities)
2. **Dependent modules** next (RoleManager, OverrideManager)
3. **Main orchestrator** last (content.js)

❌ INCORRECT Order:

- Dependent modules before base modules
- Main script before all modules

🔍 Debugging Module Issues

javascript

```
// Debug snippet to check module loading
console.log('=== MODULE DEBUG INFO ===');
console.log('Window modules:', Object.keys(window).filter(k => k.startsWith('ePTW_')));
console.log('Document readyState:', document.readyState);
console.log('========================');

// Add to content.js for timing issues
await new Promise(resolve => setTimeout(resolve, 50)); // Small delay
```

🛡️ Error Handling Patterns

javascript

```
// Option A: Fail-fast (Recommended)
if (typeof window.ePTW_AppState === 'undefined') {
    console.error('CRITICAL: AppState module not loaded');
    return; // Stop initialization
}

// Option B: Graceful degradation
if (typeof window.ePTW_FeatureManager !== 'undefined') {
    this.modules.featureManager = new window.ePTW_FeatureManager(this.appState);
} else {
    console.warn('FeatureManager not available - some features disabled');
}
```

📦 Asset Management

javascript

```
// modules/OverrideManager.js - Using pre-loaded assets
class OverrideManager {
    async downloadReadme() {
        const readmeContent = window.ePTW_ASSETS.README || await
this.loadReadmeFallback();
        await this.downloadFile(readmeContent, 'README.txt', 'text/plain');
    }

    async loadReadmeFallback() {
        const readmeUrl = chrome.runtime.getURL('assets/README_SOC_OVERRIDES.txt');
        const response = await fetch(readmeUrl);
        return await response.text();
    }
}
```

🎨 Advanced: Module Communication

javascript

```
// Event-based communication between modules
window.ePTW_EVENTS = {
    emit(event, data) {
        const customEvent = new CustomEvent(`ePTW_${event}`, { detail: data });
        window.dispatchEvent(customEvent);
    },

    on(event, callback) {
        window.addEventListener(`ePTW_${event}`, (e) => callback(e.detail));
    }
};

// Usage in modules
window.ePTW_EVENTS.emit('roleChanged', { newRole: 'Admin' });
window.ePTW_EVENTS.on('roleChanged', (data) => {
    console.log('Role changed to:', data.newRole);
});
```

📋 Quick Checklist

- Use IIFE pattern: `(function() { ... })();`
- Attach to window: `window.ePTW_ModuleName = ClassName`
- Order modules in manifest correctly
- Use dependency injection in constructors
- Pre-load assets for better performance
- Add debug logging for module loading
- Handle missing modules gracefully
- Keep main content script as orchestrator

🚀 Benefits Achieved

- **Modularity**: Each module has single responsibility
- **Maintainability**: Easy to update individual modules
- **Testability**: Modules can be tested in isolation
- **Performance**: Lazy loading of features
- **Debugging**: Clear module boundaries and responsibilities

This pattern makes your Chrome extension scalable, maintainable, and professional! 🎉