JavaScript Namespace Pattern Cheat Sheet

Basic Pattern Template

javascript

```
!(function(namespace) {
    // Your code here
    namespace.ModuleName = {
        // properties and methods
    };
}(Namespace || (Namespace = {})));
```

1. Basic Namespace Setup

Create or Extend Namespace

javascript

```
// Single file approach
!(function(common) {
    common.version = "1.0.0";
}(Common || (Common = {})));

// Multiple files can safely use the same pattern
!(function(common) {
    common.Utils = {
        // utility methods
    };
}(Common || (Common = {})));
```

2. Module Organization Examples

Utility Module

javascript

```
!(function(common) {
    common.Utils = {
        // Constants
        VERSION: "1.0.0",

        // Methods
        generateId: function() {
            return Math.random().toString(36).substr(2, 9);
        },

        formatDate: function(date) {
            return date.toISOString().split('T')[0];
        },

        deepClone: function(obj) {
            return JSON.parse(JSON.stringify(obj));
        }
    };
}(Common || (Common = {})));
```

API Module

javascript

```
!(function(common) {
    common.API = {
        baseURL: "https://api.example.com",

        // HTTP methods
        get: function(endpoint) {
            return fetch(`${this.baseURL}${endpoint}`).then(r => r.json());
        },

        post: function(endpoint, data) {
            return fetch(`${this.baseURL}${endpoint}`, {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify(data)
            }).then(r => r.json());
        },

        // Specific API calls
        users: {
            getAll: function() { return common.API.get('/users'); },
            getById: function(id) { return common.API.get(`/users/${id}`); },
            create: function(userData) { return common.API.post('/users', userData); }
        }
    };
}(Common || (Common = {})));
```

UI Components Module

javascript

```
!(function(common) {
    common.UI = {
        // DOM utilities
        $: function(selector) {
            return document.querySelector(selector);
        },

        show: function(element) {
            element.style.display = 'block';
        },

        hide: function(element) {
            element.style.display = 'none';
        },

        // Component-specific logic
        Modal: {
            open: function(modalId) {
                const modal = common.UI.$(`#${modalId}`);
                common.UI.show(modal);
            },
            close: function(modalId) {
                const modal = common.UI.$(`#${modalId}`);
                common.UI.hide(modal);
            }
        }
    };
}(Common || (Common = {})));
```

3. Class-Based Organization

Using Constructor Functions

javascript

```
!(function(common) {
    // User class
    common.User = function(name, email) {
        this.name = name;
        this.email = email;
        this.id = common.Utils.generateId();
    };

    common.User.prototype = {
        getName: function() { return this.name; },
        getEmail: function() { return this.email; },
        toJSON: function() {
            return { name: this.name, email: this.email, id: this.id };
        }
    };

    // User manager
    common.UserManager = {
        users: [],

        addUser: function(name, email) {
            const user = new common.User(name, email);
            this.users.push(user);
            return user;
        },

        findUser: function(id) {
            return this.users.find(user => user.id === id);
        },

        getAllUsers: function() {
            return this.users.map(user => user.toJSON());
        }
    };
}(Common || (Common = {})));
```

4. Advanced Patterns

Private Variables with Closure

javascript

```
!(function(common) {
    var privateCounter = 0; // Private variable

    common.Counter = {
        increment: function() {
            privateCounter++;
            return privateCounter;
        },

        getCount: function() {
            return privateCounter;
        },

        reset: function() {
            privateCounter = 0;
        }
    };
}(Common || (Common = {})));
```

Event System

javascript

```
!(function(common) {
    var events = {}; // Private event registry

    common.Events = {
        on: function(eventName, callback) {
            if (!events[eventName]) events[eventName] = [];
            events[eventName].push(callback);
        },

        off: function(eventName, callback) {
            if (events[eventName]) {
                events[eventName] = events[eventName].filter(cb => cb !== callback);
            }
        },

        emit: function(eventName, data) {
            if (events[eventName]) {
                events[eventName].forEach(callback => callback(data));
            }
        }
    };
}(Common || (Common = {})));
```

5. Usage Examples

Basic Usage

javascript

```
// After including all module files
console.log(Common.Utils.generateId()); // "a1b2c3d4e"
console.log(Common.version); // "1.0.0"

// Using API module
Common.API.users.getAll().then(users => {
    console.log(users);
});

// Using UI components
Common.UI.Modal.open('user-modal');
```

Chaining Modules

javascript

```
// Create a user and save via API
const user = Common.UserManager.addUser('John', 'john@example.com');
Common.API.users.create(user.toJSON()).then(result => {
    Common.Events.emit('user:created', result);
});
```

6. File Organization

Recommended File Structure

text

```
js/
├── common-namespace.js      // Core namespace setup
├── common-utils.js          // Utility functions
├── common-api.js           // API interactions
├── common-ui.js            // UI components
├── common-events.js        // Event system
└── app.js                  // Main application code
```

Each File Pattern

javascript

```
// common-utils.js
!(function(common) {
    common.Utils = {
        // Utility methods...
    };
}(Common || (Common = {})));

// common-api.js
!(function(common) {
    common.API = {
        // API methods...
    };
}(Common || (Common = {})));
```

7. Benefits Summary

- ✅ **No Global Pollution** - Only `Common` is in global scope
- ✅ **Safe Multiple File Loading** - Files can load in any order
- ✅ **Modular Organization** - Logical separation of concerns
- ✅ **Private Variables** - Closure protects internal state
- ✅ **Minification Friendly** - Pattern survives minification
- ✅ **Progressive Enhancement** - Add modules over time

Quick Reference Card

javascript

```javascript
// Basic Template
!(function(ns) { ns.Module = {}; }(Namespace || (Namespace = {})));

// With private variables
!(function(ns) { var private = 0; ns.Module = {}; }(Common || (Common = {})));

// With dependencies
!(function(ns) {
    ns.Module = {
        method: function() { ns.Utils.helper(); }
    };
}(Common || (Common = {})));
```