Programming Reference

Building with LARC

Modern Web Architecture Using Open Standards.

Christopher Robison

Building with LARC: A Reference Manual

A Comprehensive Guide to Modern Web Applications

LARC Project Contributors

December 2025

1 Introduction

Building with LARC is a comprehensive reference manual for the Lightweight Asynchronous Relay Core (LARC) framework. This book documents LARC's architecture, components, APIs, and patterns for experienced developers who need detailed technical reference material.

1.1 Prerequisites

This is a reference manual, not a tutorial. Before using this book, you should:

1. **Read *Learning LARC* first** — The companion tutorial book teaches LARC concepts, architecture, and development patterns through hands-on examples.
2. **Have web development experience** — Familiarity with JavaScript (ES6+), HTML5, CSS3, Web Components, HTTP/REST APIs, and browser development tools.
3. **Understand LARC basics** — Core concepts (PAN bus, pub/sub messaging, component lifecycle) covered in *Learning LARC* Chapters 1-5.

If you're new to LARC, start with *Learning LARC*. This reference assumes you already know how to build LARC applications and need API documentation or implementation details.

1.2 How to Use This Book

1.2.1 As a Reference

Jump directly to chapters covering your current need:

- **Quick lookup**: Use Part I (Chapters 1-3) for architecture overview and configuration
- **Task-specific**: Part II (Chapters 4-16) covers specific features (state, routing, forms, etc.)
- **Component APIs**: Part III (Chapters 17-21) provides exhaustive component documentation
- **Quick reference**: Part IV (Appendices A-G) for message topics, patterns, glossary

1.2.2 Book Structure

Part I: Quick Reference (Chapters 1-3) - Architecture overview, message patterns, configuration options

Part II: Feature Implementation (Chapters 4-16) - State management, routing, forms, APIs, authentication, real-time features, file management, theming, performance, testing, debugging, patterns, deployment

Part III: Component Reference (Chapters 17-21) - Complete API documentation for core, data, UI, integration, and utility components

Part IV: Appendices (A-G) - Message topics, event envelope spec, configuration, migration guide, recipes, glossary, resources

1.3 Conventions

1.3.1 Code Examples

Inline code: `component-name`, `attribute="value"`, `methodName()`

Code blocks:

```
// JavaScript examples
class MyComponent extends HTMLElement {
  connectedCallback() {
    this.render();
  }
}
```

```
<!-- HTML examples -->
<pan-card title="Example">
  <p>Content</p>
</pan-card>
```

Command line:

```
$ npm install @larcjs/core
$ python3 -m http.server 8000
```

The `$` is the shell prompt — don't type it.

1.3.2 Typography

- **Bold**: New terms, emphasis, UI elements
- *Italic*: File names, URLs, emphasis
- `Constant width`: Code, commands, components, attributes

1.3.3 Annotations

> ***NOTE**: Additional context or clarification*

> ***WARNING**: Common mistakes or surprising behavior*

> ***TIP**: Practical advice from experience*

1.3.4 Message Topics

LARC uses hierarchical dot notation:

```
namespace.entity.action
```

Examples:

- `user.auth.login` — User login event
- `cart.items.add` — Add cart item
- `user.*` — Wildcard: all user events

See Appendix A for complete topic registry.

1.3.5 Component Naming

- All lowercase with hyphens: `pan-button`, `pan-card`
- Core components use `pan-` prefix
- Custom components use your own prefix or none

1.3.6 API Signatures

Type annotations for clarity (not actual TypeScript):

```
publish(topic: string, payload: any, options?: object): void
```

TypeScript users: See `@larcjs/core-types` for official definitions.

1.4 Relationship to *Learning LARC*

	Learning LARC	Building with LARC
Purpose	Tutorial	Reference
Audience	Beginners	Experienced developers
Style	Narrative, progressive	Dense, lookup-focused
Coverage	Curated essentials	Comprehensive
When to use	Learning concepts	Building applications

New to LARC? Complete *Learning LARC* before using this reference.

Experienced? This book is your primary resource.

Learning specific features? Check *Learning LARC* for tutorials, then this book for complete details.

1.5 What's Next

- **Chapter 2**: Architecture overview and core concepts reference
- **Chapter 3**: Getting started — quick installation and setup
- **Chapters 4-16**: Task-specific implementation guides
- **Chapters 17-21**: Complete component API reference
- **Appendices**: Quick lookup tables and reference material

All examples available at `github.com/larcjs/building-with-larc-examples`

2 Core Concepts Quick Reference

This chapter provides quick lookup for LARC's core concepts. For tutorials and detailed explanations, see *Learning LARC* Chapters 2-5.

2.1 Architecture Overview

LARC applications consist of:

1. **PAN Bus** (`<pan-bus>`) — Message routing system
2. **Components** — Web Components that publish/subscribe to messages
3. **PanClient** — Helper library for working with the bus
4. **Message Topics** — Hierarchical namespaces for routing

```
<!DOCTYPE html>
<html>
<body>
  <pan-bus></pan-bus>
  <my-app></my-app>

  <script type="module" src="/app.mjs"></script>
</body>
</html>
```

2.2 Message Bus

What: DOM-based pub/sub system for component communication **Element**: `<pan-bus>` **Tutorial**: *Learning LARC* Chapter 5 **Reference**: Chapter 17

2.2.1 Basic Usage

```
import { PanClient } from '@larc/core';

const client = new PanClient();
await client.ready();

// Publish
client.publish({
  topic: 'user.login',
  data: { userId: 123 }
});

// Subscribe
client.subscribe('user.login', (msg) => {
  console.log('User logged in:', msg.data.userId);
});
```

2.2.2 Configuration

Attribute	Type	Default	Description
`max-retained`	number	1000	Maximum retained messages
`max-message-size`	number	1MB	Maximum message size in bytes
`debug`	boolean	false	Enable debug logging
`allow-global-wildcard`	boolean	true	Allow `*` wildcard subscriptions

See Chapter 17 for complete pan-bus documentation.

2.3 Pub/Sub Pattern

What: Decoupled component communication via topics **Tutorial**: *Learning LARC* Chapter 5

2.3.1 Message Flow

```
Publisher → PAN Bus → Subscribers
```

1. Component publishes message to topic
2. Bus routes message to all subscribers of that topic
3. Subscribers receive message and react

2.3.2 Wildcards

Pattern	Matches	Example
`user.*`	All user events	`user.login`, `user.logout`
`*.error`	All error events	`api.error`, `db.error`
`*`	All events	Everything (use sparingly)

2.4 Message Topics

Convention: `namespace.entity.action`

2.4.1 Examples

```
user.auth.login
user.profile.update
cart.items.add
cart.checkout.start
api.users.error
theme.changed
```

2.4.2 Best Practices

- Use lowercase with dots as separators
- Past tense for events: `user.logged-in` not `user.login`
- Present for commands: `cart.checkout.start`
- Namespace by domain: `api.*`, `user.*`, `cart.*`

See Appendix A for complete topic registry.

2.5 Components

What: Web Components that use PAN bus for communication **Tutorial**: *Learning LARC* Chapter 4 **Reference**: Chapters 17-21

2.5.1 Basic Component

```
class MyComponent extends HTMLElement {
  connectedCallback() {
    this.client = new PanClient(this);

    // Subscribe to messages
    this.client.subscribe('data.updated', (msg) => {
      this.render(msg.data);
    });

    // Publish messages
    this.dispatchAction();
  }

  dispatchAction() {
    this.client.publish({
      topic: 'action.triggered',
      data: { componentId: this.id }
    });
  }

  disconnectedCallback() {
    // PanClient automatically cleans up subscriptions
  }
}

customElements.define('my-component', MyComponent);
```

2.5.2 Lifecycle

Callback	When Called	Use For
`constructor()`	Element created	Initialize properties
`connectedCallback()`	Added to DOM	Subscribe, render, fetch data
`disconnectedCallback()`	Removed from DOM	Cleanup (automatic with PanClient)
`attributeChangedCallback()`	Attribute changes	Re-render on attribute updates

2.6 PanClient

What: Helper library for working with PAN bus **Import**: `import { PanClient } from '@larc/core'` **Tutorial**: *Learning LARC* Chapter 5 **Reference**: Chapter 17

2.6.1 API Summary

```
const client = new PanClient(element);

// Wait for bus ready
await client.ready();

// Publish message
client.publish({ topic, data, ...options });

// Subscribe to topic
client.subscribe(topic, handler);

// Request/response pattern
const response = await client.request(topic, data);

// Respond to requests
client.respond(topic, handler);

// Unsubscribe (usually automatic)
const unsubscribe = client.subscribe(topic, handler);
unsubscribe();
```

2.6.2 Constructor Options

```
new PanClient(element, options)
```

Option	Type	Default	Description
`element`	HTMLElement	null	Owner element (auto-cleanup)
`timeout`	number	5000	Request timeout (ms)
`debug`	boolean	false	Enable debug logging

2.7 State Management

What: Reactive data stores for shared state **Tutorial**: *Learning LARC* Chapter 6
Reference: Chapter 18

2.7.1 Options

Approach	Use For	Component
Component-local	UI state, temp values	N/A
Shared objects	App config, settings	N/A
`pan-store`	Reactive shared state	Chapter 18
`pan-idb`	Persistent IndexedDB	Chapter 18
LocalStorage	Simple persistence	N/A

2.8 Routing

What: URL-based navigation **Tutorial**: *Learning LARC* Chapter 9 **Reference**: Chapter 18

```
// Message-based routing
client.publish({
  topic: 'app.navigate',
  data: { path: '/users/123' }
});
```

See Chapter 18 for `pan-router` component documentation.

2.9 Forms

What: Form handling and validation **Tutorial**: *Learning LARC* Chapter 10 **Reference**: Chapter 19

```
<pan-form submit-topic="form.submitted">
  <pan-input name="email" type="email" required></pan-input>
  <pan-button type="submit">Submit</pan-button>
</pan-form>
```

See Chapter 19 for form component documentation.

2.10 Data Fetching

What: API integration patterns **Tutorial**: *Learning LARC* Chapter 11 **Reference**:

Chapter 20

```
// Via PanClient
const response = await client.request('api.users.list', {});

// Via pan-fetch component
client.publish({
  topic: 'api.fetch',
  data: {
    url: '/api/users',
    method: 'GET'
  }
});
```

See Chapter 20 for integration components.

2.11 ES Modules

LARC uses standard ES modules:

```
// Import from CDN
import { PanClient } from 'https://cdn.jsdelivr.net/npm/@larcjs/
        core/pan-client.mjs';

// Import from local
import { PanClient } from '/core/pan-client.mjs';

// Use import maps (recommended)
<script type="importmap">
{
  "imports": {
    "@larc/core": "/core/pan-client.mjs"
  }
}
</script>
```

2.12 Cross-References

- **Tutorials**: *Learning LARC* Chapters 2-5
- **PAN Bus API**: Chapter 17 (pan-bus)
- **Component Reference**: Chapters 17-21
- **Message Topics**: Appendix A
- **Configuration**: Appendix C
- **Patterns & Recipes**: Appendix E

2.13 Quick Start

```
# Install
npm install @larcjs/core

# Or use CDN
https://cdn.jsdelivr.net/npm/@larcjs/core@latest/pan-bus.mjs
```

```
<!DOCTYPE html>
<html>
<head>
  <script type="importmap">
  {
    "imports": {
      "@larc/core": "https://cdn.jsdelivr.net/npm/@larcjs/
        core@latest/pan-client.mjs"
    }
  }
  </script>
</head>
<body>
  <pan-bus debug="true"></pan-bus>
  <my-app></my-app>

  <script type="module">
    import { PanClient } from '@larc/core';

    class MyApp extends HTMLElement {
      connectedCallback() {
        this.client = new PanClient(this);
        this.innerHTML = '<h1>Hello LARC!</h1>';
      }
    }
    customElements.define('my-app', MyApp);
  </script>
</body>
</html>
```

For detailed tutorials, see *Learning LARC* Chapter 3 (Getting Started).

3 Getting Started

Quick installation and setup reference. For detailed tutorials, see *Learning LARC* Chapter 3.

3.1 Installation

3.1.1 Via CDN (No Build Tools)

```
<!DOCTYPE html>
<html>
<head>
  <script type="importmap">
  {
    "imports": {
      "@larc/core": "https://cdn.jsdelivr.net/npm/@larcjs/core@2/
        pan-client.mjs",
      "@larc/ui": "https://cdn.jsdelivr.net/npm/@larcjs/ui@2/"
    }
  }
  </script>
</head>
<body>
  <pan-bus></pan-bus>
  <my-app></my-app>

  <script type="module">
    import { PanClient } from '@larc/core';

    class MyApp extends HTMLElement {
      connectedCallback() {
        this.innerHTML = '<h1>Hello LARC!</h1>';
      }
    }
    customElements.define('my-app', MyApp);
  </script>
</body>
</html>
```

3.1.2 Via npm

```
npm install @larcjs/core @larcjs/ui
```

```
// In your module
import { PanClient } from '@larcjs/core';
import '@larcjs/ui/pan-button.mjs';
```

3.1.3 Project Structure

```
my-app/
├── index.html
├── app.mjs
├── components/
|   ├── my-component.mjs
|   └── another-component.mjs
└── styles/
    └── main.css
```

3.2 Development Server

Any static file server works:

```
# Python
python3 -m http.server 8000

# Node.js
npx http-server -p 8000

# PHP
php -S localhost:8000
```

3.3 Minimal Application

```
<!-- index.html -->
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
        scale=1.0">
  <title>LARC App</title>
  <script type="importmap">
  {
    "imports": {
      "@larc/core": "https://cdn.jsdelivr.net/npm/@larcjs/core@2/
        pan-client.mjs"
    }
  }
  </script>
</head>
<body>
  <pan-bus debug="true"></pan-bus>
  <my-app></my-app>

  <script type="module" src="/app.mjs"></script>
</body>
</html>
```

```
// app.mjs
import { PanClient } from '@larc/core';

class MyApp extends HTMLElement {
  connectedCallback() {
    this.client = new PanClient(this);

    // Subscribe to events
    this.client.subscribe('app.ready', () => {
      console.log('App is ready!');
    });

    // Render
    this.innerHTML = `
      <h1>My LARC Application</h1>
      <button id="test-btn">Test PAN Bus</button>
    `;

    // Add event listener
    this.querySelector('#test-btn').addEventListener('click', () =>
        {
      this.client.publish({
        topic: 'button.clicked',
        data: { timestamp: Date.now() }
      });
    });
  }
}

customElements.define('my-app', MyApp);
```

3.4 Import Maps

Define module aliases for cleaner imports:

```
<script type="importmap">
{
  "imports": {
    "@larc/core": "https://cdn.jsdelivr.net/npm/@larcjs/core@2/pan-
        client.mjs",
    "@larc/ui": "https://cdn.jsdelivr.net/npm/@larcjs/ui@2/",
    "@/": "./src/",
    "components/": "./components/"
  }
}
</script>
```

Then use:

```
import { PanClient } from '@larc/core';
import '@larc/ui/pan-button.mjs';
import MyComponent from '@/my-component.mjs';
import Header from 'components/header.mjs';
```

3.5 Browser Requirements

Browser	Minimum Version	Notes
Chrome	90+	Full support
Firefox	88+	Full support
Safari	14+	Full support
Edge	90+	Full support

Required features: - Custom Elements v1 - Shadow DOM v1 - ES Modules - Import Maps

Check support: `caniuse.com/custom-elementsv1`

3.6 IDE Setup

3.6.1 VS Code (Recommended)

Install extensions:

- **LARC Extension** — Snippets, IntelliSense
- **ESLint** — Code linting
- **Prettier** — Code formatting
- **Live Server** — Development server

3.6.2 TypeScript Support

```
npm install --save-dev @larcjs/core-types
```

```
// my-component.ts
import { PanClient } from '@larcjs/core';
import type { Message } from '@larcjs/core-types';

class MyComponent extends HTMLElement {
  private client: PanClient;

  connectedCallback(): void {
    this.client = new PanClient(this);

    this.client.subscribe('data.updated', (msg: Message) => {
      console.log(msg.data);
    });
  }
}
```

3.7 Quick Verification

Test your setup:

```
// Open browser console
import { PanClient } from '@larc/core';

const client = new PanClient();
await client.ready();

client.subscribe('test', (msg) => console.log('Received:',
        msg.data));
client.publish({ topic: 'test', data: { hello: 'world' } });

// Should log: Received: { hello: 'world' }
```

3.8 Next Steps

- **Tutorial**: *Learning LARC* Chapter 3 for step-by-step guide
- **Core concepts**: Chapter 2 for quick reference
- **Components**: Chapters 17-21 for API documentation
- **Examples**: `github.com/larcjs/examples`

3.9 Common Issues

3.9.1 Import Maps Not Working

Problem: Imports fail with “Failed to resolve module specifier” **Solution**: Ensure import maps are in `<head>` before any module scripts

3.9.2 PAN Bus Not Ready

Problem: `client.ready()` never resolves **Solution**: Verify `<pan-bus>` element is in DOM before creating PanClient

3.9.3 CORS Errors with CDN

Problem: Module loading fails with CORS error **Solution**: Use a local development server, not `file://` protocol

See *Learning LARC* Chapter 3 for detailed troubleshooting.

4 State Management

Quick reference for state management patterns in LARC applications. For detailed tutorials, see *Learning LARC* Chapter 6.

4.1 Overview

State management is the practice of tracking application data across components. LARC distinguishes between **local state** (component-specific) and **shared state** (application-wide), using the PAN bus for state synchronization.

Key Concepts:

- Local state: Component properties, ephemeral
- Shared state: Store components, persisted via localStorage, IndexedDB, or OPFS
- State stores: Components that manage and publish shared state
- Synchronization: Optimistic updates, debouncing, polling, conflict resolution

4.2 Quick Example

```
// State store component
class UserStore extends HTMLElement {
  constructor() {
    super();
    this.currentUser = null;
  }

  connectedCallback() {
    this.subscriptions = [
      subscribe('auth.login.success', (msg) =>
        this.setUser(msg.data)),
      subscribe('auth.logout', () => this.setUser(null))
    ];

    this.loadPersistedUser();
  }

  setUser(user) {
    this.currentUser = user;
    publish('user.current', user);

    if (user) {
      localStorage.setItem('currentUser', JSON.stringify(user));
    } else {
      localStorage.removeItem('currentUser');
    }
  }

  loadPersistedUser() {
    const stored = localStorage.getItem('currentUser');
    if (stored) {
      try {
```

```
        this.setUser(JSON.parse(stored));
      } catch (error) {
        console.error('Failed to load user:', error);
      }
    }
  }

  disconnectedCallback() {
    this.subscriptions.forEach(unsub => unsub());
  }
}

customElements.define('user-store', UserStore);
```

4.3 Persistence Strategies

Strategy	Size Limit	API Style	Use Case
localStorage	5-10 MB	Synchronous	Small settings, simple data
IndexedDB	100s of MB	Async (Promise)	Structured data, queries
OPFS	GB+	Async (File API)	Large files, binary data

4.3.1 When to Use Each

- **localStorage**: Settings, themes, small JSON (< 100 KB)

- **IndexedDB**: Documents, cached API responses, structured records
- **OPFS**: Images, videos, large text files, application data files

4.4 State Synchronization Patterns

4.4.1 Optimistic Updates

Step	Action
1	Update local state immediately
2	Publish updated state to UI
3	Sync to server in background
4	On success: Publish confirmation
5	On error: Rollback and publish error

```
async addTodo(todo) {
  // Step 1-2: Optimistic update
  const temp = { id: `temp-${Date.now()}`, ...todo };
  this.todos.push(temp);
  publish('todos.loaded', { todos: this.todos });

  try {
    // Step 3: Sync to server
    const response = await fetch('/api/todos', {
      method: 'POST',
      body: JSON.stringify(todo)
    });
    const saved = await response.json();

    // Step 4: Replace temp with server ID
    this.todos = this.todos.map(t => t.id === temp.id ? saved : t);
    publish('todos.loaded', { todos: this.todos });
  } catch (error) {
    // Step 5: Rollback on error
    this.todos = this.todos.filter(t => t.id !== temp.id);
    publish('todos.loaded', { todos: this.todos });
    publish('todo.error', { error: error.message });
  }
}
```

4.4.2 Debounced Sync

For high-frequency updates (e.g., text editor), debounce server sync while updating UI immediately:

```
updateContent(content) {
  this.content = content;
  publish('editor.content.updated', { content }); // Immediate UI
        update

  clearTimeout(this.syncTimer);
  this.syncTimer = setTimeout(() => {
    this.syncToServer(); // Delayed server sync
  }, 1000);
}
```

4.4.3 Polling

Poll server periodically for updates without WebSockets:

```
startPolling() {
  this.fetchNotifications();
  this.pollTimer = setInterval(() => {
    this.fetchNotifications();
  }, 30000); // Every 30 seconds
}
```

4.5 Conflict Resolution Strategies

Strategy	Description	Use Case
Last Write Wins	Most recent update overwrites	Simple apps, rare conflicts
Timestamps	Keep update with newest timestamp	Async updates, out-of-order messages
Version Vectors	Track causality across clients	Distributed systems, offline-first
User Intervention	Detect conflict, prompt user	Collaborative apps, important data

4.5.1 Example: Timestamp-Based Resolution

```
connectedCallback() {
  this.unsubscribe = subscribe('data.update', (msg) => {
    const { key, value, timestamp } = msg.data;

    // Only apply if newer
    if (!this.timestamps[key] || timestamp > this.timestamps[key]) {
      this.data[key] = value;
      this.timestamps[key] = timestamp;
      publish('data.current', this.data);
    }
  });
}
```

4.6 Derived State Pattern

Compute derived state from source state rather than storing redundantly:

```
publishDerivedState() {
  const itemCount = this.items.reduce((sum, item) => sum +
        item.quantity, 0);
  const subtotal = this.items.reduce((sum, item) => sum +
        (item.price * item.quantity), 0);
  const tax = subtotal * 0.08;
  const total = subtotal + tax;

  publish('cart.state', {
    items: this.items,
    itemCount,
    subtotal,
    tax,
    total
  });
}
```

4.7 History and Time Travel

Implement undo/redo by maintaining state snapshots:

```
addSnapshot(state) {
  this.history = this.history.slice(0, this.currentIndex + 1);
  this.history.push(JSON.parse(JSON.stringify(state)));
  this.currentIndex++;

  if (this.history.length > this.maxHistory) {
    this.history.shift();
    this.currentIndex--;
  }

  publish('state.current', state);
  publish('state.history.updated', {
    canUndo: this.currentIndex > 0,
    canRedo: this.currentIndex < this.history.length - 1
  });
}
```

4.8 Performance Best Practices

Practice	Why
Minimize updates	Only publish when state actually changes
Batch updates	Combine multiple field changes into single message
Immutable updates	Create new objects, don’t mutate
Debounce high-frequency	Don’t publish every keystroke
Lazy load	Load data on demand
Prune old data	Remove stale data to prevent memory bloat

4.9 Component Reference

- **pan-store**: Reactive state store with persistence - See Chapter 18
- **pan-idb**: IndexedDB wrapper component - See Chapter 18

4.10 Complete Example: Document Store with IndexedDB

```
class DocumentStore extends HTMLElement {
  constructor() {
    super();
    this.db = new IndexedDBStore('app-db', 'documents');
    this.documents = [];
  }

  async connectedCallback() {
    this.subscriptions = [
      subscribe('document.save', async (msg) => await
        this.saveDocument(msg.data)),
      subscribe('document.delete', async (msg) => await
        this.deleteDocument(msg.data.id)),
      subscribe('document.load', async (msg) => await
        this.loadDocument(msg.data.id))
    ];

    await this.loadAllDocuments();
  }

  async loadAllDocuments() {
    try {
      this.documents = await this.db.getAll();
      publish('documents.loaded', { documents: this.documents });
    } catch (error) {
      console.error('Failed to load documents:', error);
      publish('documents.error', { error: error.message });
    }
  }

  async saveDocument(document) {
    try {
      await this.db.put(document);
```

```
      this.documents = await this.db.getAll();
      publish('document.saved', { document });
      publish('documents.loaded', { documents: this.documents });
    } catch (error) {
      console.error('Failed to save document:', error);
      publish('document.error', { error: error.message });
    }
  }

  async deleteDocument(id) {
    try {
      await this.db.delete(id);
      this.documents = await this.db.getAll();
      publish('document.deleted', { id });
      publish('documents.loaded', { documents: this.documents });
    } catch (error) {
      console.error('Failed to delete document:', error);
      publish('document.error', { error: error.message });
    }
  }

  async loadDocument(id) {
    try {
      const document = await this.db.get(id);
      publish('document.loaded', { document });
    } catch (error) {
      console.error('Failed to load document:', error);
      publish('document.error', { error: error.message });
    }
  }

  disconnectedCallback() {
    this.subscriptions.forEach(unsub => unsub());
```

```
  }
}

customElements.define('document-store', DocumentStore);
```

4.11 Cross-References

- **Tutorial**: *Learning LARC* Chapter 6 (State Management)
- **Components**: Chapter 18 (pan-store, pan-idb)
- **Patterns**: Appendix E (Message Patterns)
- **Related**: Chapter 5 (Routing), Chapter 7 (Data Fetching)

4.12 Common Issues

4.12.1 Issue: State not persisting

Problem: Data lost on page reload **Solution**: Ensure `loadPersistedUser()` called in `connectedCallback()` and storage API used correctly

4.12.2 Issue: Race conditions

Problem: Concurrent updates causing inconsistent state **Solution**: Use timestamps or version vectors, implement conflict resolution strategy

4.12.3 Issue: Memory leaks from subscriptions

Problem: Memory grows over time **Solution**: Always unsubscribe in `disconnectedCallback()`, store subscription functions

4.12.4 Issue: localStorage quota exceeded

Problem: `QuotaExceededError` thrown **Solution**: Migrate to IndexedDB for larger data, implement data pruning strategy

4.12.5 Issue: Stale data after optimistic update failure

Problem: UI shows incorrect state after server error **Solution**: Implement rollback in catch block, publish error state

See *Learning LARC* Chapter 6 for detailed troubleshooting and advanced patterns.

5 Routing and Navigation

Quick reference for client-side routing with LARC's `pan-routes` component. For detailed tutorials, see *Learning LARC* Chapter 9.

5.1 Overview

Client-side routing updates the URL and renders components without full page reloads. LARC's `pan-routes` component matches URL patterns to components, enabling seamless navigation experiences.

Key Concepts:

- Route patterns: Map URLs to components
- Dynamic parameters: Extract values from URLs (`:id` , `:username`)
- Navigation guards: Control access to routes
- Deep linking: Encode application state in URLs
- History management: Push vs. replace navigation

5.2 Quick Example

```
<pan-app>
  <pan-routes>
    <pan-route path="/" component="home-view"></pan-route>
    <pan-route path="/products" component="product-list"></pan-
        route>
    <pan-route path="/products/:id" component="product-
        detail"></pan-route>
    <pan-route path="*" component="not-found-view"></pan-route>
  </pan-routes>
</pan-app>
```

```
class ProductDetail extends LarcComponent {
  onRoute(params) {
    // params.id extracted from URL
    this.loadProduct(params.id);
  }

  async loadProduct(id) {
    const response = await fetch(`/api/products/${id}`);
    this.product = await response.json();
    this.render();
  }
}
```

5.3 Route Patterns

5.3.1 Route Matching Order

Routes evaluate **top to bottom**. Place specific routes before general ones:

Priority	Pattern	Description
1	`/login`	Exact static paths first
2	`/products/new`	Static before dynamic
3	`/products/:id`	Dynamic parameters
4	`/admin/:section`	General patterns
5	`*`	Catch-all (404) last

5.3.2 Dynamic Parameters

Pattern	Example URL	Extracted Params
`/products/:id`	`/products/123`	`{ id: '123' }`
`/users/:username`	`/users/alice`	`{ username: 'alice' }`
`/posts/:year/:month/:slug`	`/posts/2024/12/hello`	`{ year: '2024', month: '12', slug: 'hello' }`
`/store/:category/:subcategory/:id`	`/store/electronics/phones/456`	`{ category: 'electronics', subcategory: 'phones', id: '456' }`

5.3.3 Nested Routes

Use wildcard suffix (`*`) for nested route sections:

```
<pan-routes>
  <pan-route path="/" component="home-view"></pan-route>
  <pan-route path="/products*" component="product-section"></pan-
      route>
  <pan-route path="/admin*" component="admin-section"></pan-route>
</pan-routes>
```

Product section contains its own sub-routes:

```
class ProductSection extends LarcComponent {
  template() {
    return `
      <div class="product-section">
        <nav class="sidebar">...</nav>
        <main>
          <pan-routes>
            <pan-route path="/products" component="product-
        list"></pan-route>
            <pan-route path="/products/new" component="product-
        form"></pan-route>
            <pan-route path="/products/:id" component="product-
        detail"></pan-route>
          </pan-routes>
        </main>
      </div>
    `;
  }
}
```

5.4 Programmatic Navigation

5.4.1 navigate() Function

```
import { navigate } from '@larc/core';

// Basic navigation
navigate('/products/123');

// Replace current history entry (don't add to back button)
navigate('/login', { replace: true });

// Navigate back/forward
navigate(-1);  // Back one page
navigate(1);   // Forward one page
navigate(-2);  // Back two pages

// Navigation after form submission
async handleLogin(event) {
  event.preventDefault();
  const response = await fetch('/api/login', { /*...*/ });

  if (response.ok) {
    navigate('/dashboard');
  } else {
    this.showError('Invalid credentials');
  }
}
```

5.4.2 When to Use Replace

Scenario	Use Replace
Login redirects	Yes - Skip intermediate loading pages
Fixing invalid URLs	Yes - User shouldn’t return to bad URL
Multi-step wizards	No - Allow back navigation between steps
Normal navigation	No - Default push behavior

5.5 Navigation Guards

Control access to routes based on authentication, roles, or state.

5.5.1 Auth Guard Pattern

```
class AuthGuard {
  constructor() {
    this.user = null;
    this.loadUserFromStorage();
  }

  loadUserFromStorage() {
    const stored = localStorage.getItem('user');
    if (stored) this.user = JSON.parse(stored);
  }

  canActivate(route) {
    if (!this.user) {
      navigate('/login?redirect=' + encodeURIComponent(route.path));
      return false;
    }
    return true;
  }

  requiresRole(role) {
    return this.user && this.user.roles.includes(role);
  }
}

const authGuard = new AuthGuard();
```

5.5.2 Component Guard Hooks

```
class DashboardView extends LarcComponent {
  beforeRoute(params) {
    if (!authGuard.canActivate(this.route)) {
      return false; // Cancel navigation
    }
    return true; // Allow navigation
  }

  onRoute(params) {
    this.loadDashboardData();
  }
}

class AdminPanel extends LarcComponent {
  beforeRoute(params) {
    if (!authGuard.requiresRole('admin')) {
      navigate('/unauthorized');
      return false;
    }
    return true;
  }
}

class PostEditor extends LarcComponent {
  beforeRouteLeave(to, from) {
    if (this.isDirty) {
      return confirm('You have unsaved changes. Leave anyway?');
    }
    return true;
  }
}
```

5.6 Query Parameters and Deep Linking

Encode filters, search, pagination in URLs for bookmarkable state.

```
class ProductList extends LarcComponent {
  onRoute(params, query) {
    const {
      category = 'all',
      sort = 'name',
      page = 1,
      search = ''
    } = query;

    this.loadProducts({ category, sort, page, search });
  }

  handleFilterChange(category) {
    const currentQuery = this.getQueryParams();
    navigate(`/products?${new URLSearchParams({
      ...currentQuery,
      category,
      page: 1
    })}`);
  }

  getQueryParams() {
    return Object.fromEntries(
      new URLSearchParams(window.location.search)
    );
  }
}
```

5.7 Hash Fragments

Use for section scrolling and anchors:

```
class DocumentationView extends LarcComponent {
  afterRender() {
    const hash = window.location.hash;
    if (hash) {
      const element = this.querySelector(hash);
      if (element) {
        element.scrollIntoView({ behavior: 'smooth' });
      }
    }
  }

  template() {
    return `
      <article>
        <h1 id="introduction">Introduction</h1>
        <p>Content...</p>

        <h2 id="getting-started">Getting Started</h2>
        <p>More content...</p>

        <nav>
          <a href="#introduction">Introduction</a>
          <a href="#getting-started">Getting Started</a>
        </nav>
      </article>
    `;
  }
}
```

5.8 History Management

5.8.1 Push vs. Replace

```
// Push (default): Adds to history, back button works
navigate('/products/123');

// Replace: Replaces current entry, skips this page on back
navigate('/login', { replace: true });
```

5.8.2 Scroll Position Preservation

```
class ScrollManager {
  constructor() {
    this.positions = new Map();
    this.setupListeners();
  }

  setupListeners() {
    window.addEventListener('beforeunload', () => {
      this.savePosition(window.location.pathname);
    });

    window.addEventListener('load', () => {
      this.restorePosition(window.location.pathname);
    });
  }

  savePosition(path) {
    this.positions.set(path, {
      x: window.scrollX,
      y: window.scrollY
    });
  }

  restorePosition(path) {
    const position = this.positions.get(path);
    if (position) {
      window.scrollTo(position.x, position.y);
    } else {
      window.scrollTo(0, 0);
    }
  }
}
```

5.8.3 History Event Listener

```
window.addEventListener('popstate', (event) => {
  // User clicked back/forward button
  this.handleNavigation(event.state);
});
```

5.9 Active Link States

Highlight current route in navigation:

```
class NavBar extends LarcComponent {
  constructor() {
    super();
    this.currentPath = window.location.pathname;

    window.addEventListener('popstate', () => {
      this.currentPath = window.location.pathname;
      this.render();
    });
  }

  isActive(path) {
    return this.currentPath === path;
  }

  isActivePrefix(prefix) {
    return this.currentPath.startsWith(prefix);
  }

  template() {
    return `
      <nav>
        <a href="/" class="${this.isActive('/') ? 'active' :
        ''}">Home</a>
        <a href="/products" class="${this.isActivePrefix('/products')
        ? 'active' : ''}">Products</a>
        <a href="/about" class="${this.isActive('/about') ?
        'active' : ''}">About</a>
      </nav>
    `;
  }
}
```

5.10 SEO Considerations

5.10.1 Update Meta Tags on Route Change

```
class SEOManager {
  updateMeta(route, data) {
    document.title = data.title || 'Default Title';
    this.setMetaTag('description', data.description || '');
    this.setMetaTag('og:title', data.title);
    this.setMetaTag('og:description', data.description);
    this.setMetaTag('og:url', window.location.href);
    this.setLinkTag('canonical', window.location.href);
  }

  setMetaTag(name, content) {
    let element = document.querySelector(`meta[name="${name}"]`) ||
        document.querySelector(`meta[property="${name}"]`);

    if (!element) {
      element = document.createElement('meta');
      const attr = name.startsWith('og:') ? 'property' : 'name';
      element.setAttribute(attr, name);
      document.head.appendChild(element);
    }

    element.setAttribute('content', content);
  }

  setLinkTag(rel, href) {
    let element = document.querySelector(`link[rel="${rel}"]`);
    if (!element) {
      element = document.createElement('link');
      element.setAttribute('rel', rel);
      document.head.appendChild(element);
    }
```

```
    element.setAttribute('href', href);
  }
}

const seoManager = new SEOManager();

class ProductDetail extends LarcComponent {
  async onRoute(params) {
    const product = await this.loadProduct(params.id);

    seoManager.updateMeta(this.route, {
      title: `${product.name} - Our Store`,
      description: product.description,
      image: product.imageUrl
    });
  }
}
```

5.10.2 SSR and Prerendering

For maximum SEO, consider:

- **Server-side rendering (SSR)**: Render initial HTML on server
- **Static prerendering**: Generate HTML at build time for static routes
- **Dynamic rendering**: Serve prerendered HTML to crawlers, SPA to users

See Appendix C (Deployment) for SSR and prerendering strategies.

5.11 Component Reference

See Chapter 18 for pan-routes API documentation and message patterns.

5.12 Complete Example: E-commerce App Routing

```
import { LarcComponent, navigate } from '@larc/core';

class MainApp extends LarcComponent {
  constructor() {
    super();
    this.authGuard = new AuthGuard();
    this.seoManager = new SEOManager();
    this.scrollManager = new ScrollManager();
    this.setupNavigation();
  }

  setupNavigation() {
    window.addEventListener('popstate', () => {
      this.render();
    });
  }

  template() {
    return `
      <div class="app">
        <app-header></app-header>
        <main>
          <pan-routes>
            <pan-route path="/" component="home-view"></pan-route>
            <pan-route path="/products" component="product-
        list"></pan-route>
            <pan-route path="/products/:id" component="product-
        detail"></pan-route>
            <pan-route path="/cart" component="shopping-cart"></pan-
        route>
            <pan-route path="/checkout" component="checkout-
        view"></pan-route>
            <pan-route path="/account*" component="account-
        section"></pan-route>
```

```
            <pan-route path="*" component="not-found-view"></pan-
        route>
          </pan-routes>
        </main>
        <app-footer></app-footer>
      </div>
    `;
  }
}

customElements.define('main-app', MainApp);
```

5.13 Cross-References

- **Tutorial**: *Learning LARC* Chapter 9 (Routing and Navigation)
- **Components**: Chapter 18 (pan-routes API)
- **Patterns**: Appendix E (Navigation Patterns)
- **Related**: Chapter 4 (State Management), Chapter 16 (Deployment/SEO)

5.14 Common Issues

5.14.1 Issue: Routes not matching

Problem: URL changes but component doesn't render **Solution**: Check route order (specific before general), ensure pan-routes is in DOM

5.14.2 Issue: Back button breaks app

Problem: Clicking back causes errors or blank page **Solution**: Implement proper

cleanup in `disconnectedCallback()`, save/restore scroll position

5.14.3 Issue: Parameters not updating

Problem: Component doesn't re-render when URL params change **Solution**: Implement `onRoute(params)` lifecycle hook to react to param changes

5.14.4 Issue: SEO crawlers not indexing pages

Problem: Search engines see empty content **Solution**: Implement SSR or prerendering, ensure meta tags update correctly

5.14.5 Issue: Nested routes conflict

Problem: Child routes override parent routes **Solution**: Use wildcard suffix (`*`) in parent route pattern, ensure child patterns are more specific

See *Learning LARC* Chapter 9 for detailed troubleshooting and advanced routing patterns.

6 Forms and User Input

Quick reference for form handling in LARC applications. For detailed tutorials, see *Learning LARC* Chapter 10.

6.1 Overview

Forms enable user input through validation, submission handling, and error feedback. LARC uses standard HTML forms with JavaScript event handling—no magic, no framework-specific syntax.

Key Concepts:

- FormData API: Extract values from form inputs
- Validation strategies: Client-side and server-side validation
- Loading states: Provide feedback during async operations
- Error handling: Display meaningful messages
- File uploads: Handle files with modern APIs

6.2 Quick Example

```
class LoginForm extends LarcComponent {
  constructor() {
    super();
    this.error = null;
    this.loading = false;
  }

  async handleSubmit(event) {
    event.preventDefault();

    this.loading = true;
    this.error = null;
    this.render();

    const formData = new FormData(event.target);
    const credentials = {
      email: formData.get('email'),
      password: formData.get('password')
    };

    try {
      const response = await fetch('/api/login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(credentials)
      });

      if (response.ok) {
        const user = await response.json();
        localStorage.setItem('user', JSON.stringify(user));
        navigate('/dashboard');
      } else {
```

```
        this.error = 'Invalid credentials';
      }
    } catch (err) {
      this.error = 'Network error. Please try again.';
    } finally {
      this.loading = false;
      this.render();
    }
  }

  template() {
    return `
      <form onsubmit="this.handleSubmit(event)">
        ${this.error ? `<div class="error">${this.error}</div>` :
        ''}

        <input type="email" name="email" required
        autocomplete="email">
        <input type="password" name="password" required>

        <button type="submit" ?disabled="${this.loading}">
          ${this.loading ? 'Logging in...' : 'Login'}
        </button>
      </form>
    `;
  }
}
```

6.3 Validation Strategies

Strategy	When	Implementation
HTML5 validation	Simple cases	Use `required`, `pattern`, `min`, `max` attributes
Client-side JS	Immediate feedback	Validate on `input` or `blur` events
Server-side	Final authority	Always validate on server, return errors
Schema validation	Complex forms	Use libraries like Zod, Yup, or JSON Schema

6.3.1 Validation Example

```
validateEmail(email) {
  const pattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  return pattern.test(email);
}

handleInput(event) {
  const email = event.target.value;
  const isValid = this.validateEmail(email);

  const errorEl = this.querySelector('.email-error');
  errorEl.textContent = isValid ? '' : 'Invalid email format';
  errorEl.hidden = isValid;
}
```

6.4 Common Patterns

6.4.1 Pattern 1: Two-Way Data Binding

```
class UserProfile extends LarcComponent {
  constructor() {
    super();
    this.user = { name: '', email: '', bio: '' };
  }

  handleInput(field, event) {
    this.user[field] = event.target.value;
    this.updatePreview();
  }

  template() {
    return `
      <form>
        <input type="text" value="${this.user.name}"
               oninput="this.handleInput('name', event)">
        <textarea oninput="this.handleInput('bio',
        event)">${this.user.bio}</textarea>
        <div class="preview">${this.user.bio || 'No bio'}</div>
      </form>
    `;
  }
}
```

6.4.2 Pattern 2: File Upload

```
async handleFileUpload(event) {
  const file = event.target.files[0];
  if (!file) return;

  const formData = new FormData();
  formData.append('file', file);

  try {
    const response = await fetch('/api/upload', {
      method: 'POST',
      body: formData
    });

    if (response.ok) {
      const result = await response.json();
      this.handleUploadSuccess(result);
    }
  } catch (err) {
    this.showError('Upload failed');
  }
}
```

6.4.3 Pattern 3: Schema-Driven Validation

```
import { z } from 'zod';

const userSchema = z.object({
  email: z.string().email(),
  password: z.string().min(8),
  age: z.number().min(13).max(120)
});

async handleSubmit(event) {
  event.preventDefault();

  const data = {
    email: formData.get('email'),
    password: formData.get('password'),
    age: parseInt(formData.get('age'))
  };

  try {
    const validated = userSchema.parse(data);
    await this.submitForm(validated);
  } catch (err) {
    this.displayValidationErrors(err.errors);
  }
}
```

6.5 Input Types Reference

Type	Use Case	Validation
`text`	General text input	`pattern`, `minlength`, `maxlength`
`email`	Email addresses	Built-in email validation
`password`	Passwords	`minlength`, autocomplete hints
`number`	Numeric input	`min`, `max`, `step`
`tel`	Phone numbers	`pattern` for format
`url`	URLs	Built-in URL validation
`date`	Dates	`min`, `max` for range
`file`	File upload	`accept` for file types
`checkbox`	Boolean choice	Check `.checked` property
`radio`	Single choice	Group by `name` attribute
`select`	Dropdown choice	Multiple with `multiple` attribute

6.6 Error Display Patterns

6.6.1 Inline Errors

```
template() {
  return `
    <div class="form-group">
      <label for="email">Email</label>
      <input type="email" id="email" name="email">
      <span class="error" ?hidden="${!this.errors.email}">
        ${this.errors.email}
      </span>
    </div>
  `;
}
```

6.6.2 Summary Errors

```
template() {
  return `
    ${this.errors.length > 0 ? `
      <div class="error-summary">
        <h3>Please fix the following errors:</h3>
        <ul>
          ${this.errors.map(err => `<li>${err}</li>`).join('')}
        </ul>
      </div>
    ` : ''}
    <form>...</form>
  `;
}
```

6.7 Component Reference

- **pan-files**: File upload component - See Chapter 19
- **pan-markdown-editor**: Rich text editing - See Chapter 19

6.8 Complete Example: Registration Form

```
class RegistrationForm extends LarcComponent {
  constructor() {
    super();
    this.errors = {};
    this.loading = false;
  }

  validate(data) {
    const errors = {};

    if (!data.email || !this.validateEmail(data.email)) {
      errors.email = 'Valid email required';
    }

    if (!data.password || data.password.length < 8) {
      errors.password = 'Password must be at least 8 characters';
    }

    if (data.password !== data.confirmPassword) {
      errors.confirmPassword = 'Passwords must match';
    }

    return Object.keys(errors).length === 0 ? null : errors;
  }

  async handleSubmit(event) {
    event.preventDefault();

    const formData = new FormData(event.target);
    const data = {
      email: formData.get('email'),
      password: formData.get('password'),
```

```
      confirmPassword: formData.get('confirmPassword'),
      terms: formData.get('terms') === 'on'
    };

    // Client-side validation
    const validationErrors = this.validate(data);
    if (validationErrors) {
      this.errors = validationErrors;
      this.render();
      return;
    }

    // Submit to server
    this.loading = true;
    this.render();

    try {
      const response = await fetch('/api/register', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ email: data.email, password:
        data.password })
      });

      if (response.ok) {
        navigate('/welcome');
      } else {
        const error = await response.json();
        this.errors = { general: error.message };
      }
    } catch (err) {
      this.errors = { general: 'Network error. Please try again.' };
    } finally {
```

```
      this.loading = false;
      this.render();
    }
  }

  template() {
    return `
      <form class="registration-form"
        onsubmit="this.handleSubmit(event)">
        <h2>Create Account</h2>

        ${this.errors.general ? `
          <div class="error-banner">${this.errors.general}</div>
        ` : ''}

        <div class="form-group">
          <label for="email">Email</label>
          <input type="email" id="email" name="email" required>
          ${this.errors.email ? `<span
        class="error">${this.errors.email}</span>` : ''}
        </div>

        <div class="form-group">
          <label for="password">Password</label>
          <input type="password" id="password" name="password"
        required>
          ${this.errors.password ? `<span
        class="error">${this.errors.password}</span>` : ''}
        </div>

        <div class="form-group">
          <label for="confirmPassword">Confirm Password</label>
          <input type="password" id="confirmPassword"
        name="confirmPassword" required>
```

```
          ${this.errors.confirmPassword ? `<span
        class="error">${this.errors.confirmPassword}</span>` : ''}
        </div>

        <div class="form-group">
          <label>
            <input type="checkbox" name="terms" required>
            I agree to the Terms of Service
          </label>
        </div>

        <button type="submit" ?disabled="${this.loading}">
          ${this.loading ? 'Creating account...' : 'Sign Up'}
        </button>
      </form>
    `;
  }
}

customElements.define('registration-form', RegistrationForm);
```

6.9 Cross-References

- **Tutorial**: *Learning LARC* Chapter 10 (Forms and User Input)
- **Components**: Chapter 19 (pan-files, pan-markdown-editor)
- **Patterns**: Appendix E (Form Patterns)
- **Related**: Chapter 4 (State Management), Chapter 14 (Error Handling)

6.10 Common Issues

6.10.1 Issue: Form submits on Enter key

Problem: Single-line inputs trigger form submission **Solution**: Use `onsubmit` handler with `preventDefault()`, not button click handlers

6.10.2 Issue: FormData missing checkbox values

Problem: Unchecked checkboxes don't appear in FormData **Solution**: Check for `null` or use `.get('field') === 'on'` pattern

6.10.3 Issue: File upload fails with large files

Problem: Request timeout or server rejects large payloads **Solution**: Implement chunked upload or use signed URLs for direct S3 upload

6.10.4 Issue: Validation errors not clearing

Problem: Error messages persist after fixing input **Solution**: Clear errors object before revalidation, or validate on input event

6.10.5 Issue: Password managers not filling forms

Problem: Autocomplete not working correctly **Solution**: Use proper `autocomplete` attributes (`email`, `current-password`, `new-password`)

See *Learning LARC* Chapter 10 for detailed form patterns and advanced validation strategies.

7 Data Fetching and APIs

Quick reference for API integration and data fetching in LARC applications. For detailed tutorials, see *Learning LARC* Chapter 11.

7.1 Overview

Modern web applications fetch data from APIs using REST, GraphQL, or WebSockets. LARC provides components and patterns for handling async data loading, caching, error recovery, and real-time updates.

Key Concepts:

- REST APIs: Standard HTTP methods (GET, POST, PUT, DELETE)
- GraphQL: Query-based data fetching with precise field selection
- Error handling: Retry logic, fallbacks, user feedback
- Caching strategies: In-memory, localStorage, IndexedDB
- Loading states: Skeleton screens, spinners, optimistic updates

7.2 Quick Example

```
class ProductList extends LarcComponent {
  constructor() {
    super();
    this.products = [];
    this.loading = true;
    this.error = null;
  }

  async connectedCallback() {
    await this.loadProducts();
  }

  async loadProducts() {
    this.loading = true;
    this.error = null;
    this.render();

    try {
      const response = await fetch('/api/products');
      if (!response.ok) throw new Error('Failed to load products');

      this.products = await response.json();
    } catch (err) {
      this.error = err.message;
    } finally {
      this.loading = false;
      this.render();
    }
  }

  template() {
```

```
    if (this.loading) return '<div
        class="spinner">Loading...</div>';
    if (this.error) return `<div class="error">${this.error}</div>`;

    return `
      <div class="product-list">
        ${this.products.map(p => `
          <div class="product-card">
            <h3>${p.name}</h3>
            <p>${p.price}</p>
          </div>
        `).join('')}
      </div>
    `;
  }
}
```

7.3 REST API Patterns

HTTP Method	Purpose	Example
GET	Fetch data	`GET /api/products?category=electronics`
POST	Create resource	`POST /api/products` + body
PUT	Replace resource	`PUT /api/products/123` + body
PATCH	Update fields	`PATCH /api/products/123` + partial body
DELETE	Remove resource	`DELETE /api/products/123`

7.3.1 REST Example with Error Handling

```
async createProduct(data) {
  try {
    const response = await fetch('/api/products', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(data)
    });

    if (!response.ok) {
      const error = await response.json();
      throw new Error(error.message || 'Failed to create product');
    }

    return await response.json();
  } catch (err) {
    console.error('Create product failed:', err);
    throw err;
  }
}
```

7.4 GraphQL Integration

7.4.1 Basic Query

```
async fetchUser(userId) {
  const query = `
    query GetUser($id: ID!) {
      user(id: $id) {
        id
        name
        email
        posts {
          id
          title
        }
      }
    }
  `;

  const response = await fetch('/graphql', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      query,
      variables: { id: userId }
    })
  });

  const result = await response.json();
  if (result.errors) throw new Error(result.errors[0].message);

  return result.data.user;
}
```

7.4.2 Mutation Example

```
async updateUser(userId, updates) {
  const mutation = `
    mutation UpdateUser($id: ID!, $input: UserInput!) {
      updateUser(id: $id, input: $input) {
        id
        name
        email
      }
    }
  `;

  const response = await fetch('/graphql', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      query: mutation,
      variables: { id: userId, input: updates }
    })
  });

  const result = await response.json();
  if (result.errors) throw new Error(result.errors[0].message);

  return result.data.updateUser;
}
```

7.5 Caching Strategies

Strategy	Implementation	Use Case
In-memory	Store in component property	Session-only data
localStorage	`localStorage.setItem()`	Small datasets, settings
IndexedDB	`pan-idb` component	Large datasets, offline support
HTTP cache	`Cache-Control` headers	Static assets, CDN content
Stale-while-revalidate	Show cached + fetch fresh	Balance speed + freshness

7.5.1 Stale-While-Revalidate Pattern

```
async loadProducts() {
  // Show cached data immediately
  const cached = this.getCache('products');
  if (cached) {
    this.products = cached;
    this.render();
  }

  // Fetch fresh data in background
  try {
    const response = await fetch('/api/products');
    const fresh = await response.json();

    this.products = fresh;
    this.setCache('products', fresh, 5 * 60 * 1000); // 5 min TTL
    this.render();
  } catch (err) {
    // Keep showing cached data on error
    if (!cached) {
      this.error = err.message;
      this.render();
    }
  }
}

getCache(key) {
  const item = localStorage.getItem(`cache:${key}`);
  if (!item) return null;

  const { data, expires } = JSON.parse(item);
  if (Date.now() > expires) return null;
```

```
  return data;
}

setCache(key, data, ttl) {
  localStorage.setItem(`cache:${key}`, JSON.stringify({
    data,
    expires: Date.now() + ttl
  }));
}
```

7.6 Error Recovery Patterns

7.6.1 Retry with Exponential Backoff

```
async fetchWithRetry(url, options = {}, maxRetries = 3) {
  for (let i = 0; i < maxRetries; i++) {
    try {
      const response = await fetch(url, options);
      if (response.ok) return await response.json();

      if (response.status >= 500 && i < maxRetries - 1) {
        // Retry on server errors
        const delay = Math.pow(2, i) * 1000; // 1s, 2s, 4s
        await new Promise(resolve => setTimeout(resolve, delay));
        continue;
      }

      throw new Error(`HTTP ${response.status}`);
    } catch (err) {
      if (i === maxRetries - 1) throw err;

      const delay = Math.pow(2, i) * 1000;
      await new Promise(resolve => setTimeout(resolve, delay));
    }
  }
}
```

7.6.2 Fallback Data

```
async loadProducts() {
  try {
    const response = await fetch('/api/products');
    this.products = await response.json();
  } catch (err) {
    console.error('API failed, using fallback data:', err);
    this.products = this.getFallbackProducts();
  }

  this.render();
}

getFallbackProducts() {
  return [
    { id: 1, name: 'Product 1', price: 19.99 },
    { id: 2, name: 'Product 2', price: 29.99 }
  ];
}
```

7.7 Loading States

7.7.1 Skeleton Screens

```
template() {
  if (this.loading) {
    return `
      <div class="product-list">
        ${Array(6).fill(0).map(() => `
          <div class="product-card skeleton">
            <div class="skeleton-title"></div>
            <div class="skeleton-text"></div>
            <div class="skeleton-price"></div>
          </div>
        `).join('')}
      </div>
    `;
  }

  return this.renderProducts();
}
```

7.7.2 Progressive Loading

```
async connectedCallback() {
  // Load critical data first
  await this.loadSummary();
  this.render();

  // Load details in background
  await this.loadDetails();
  this.render();
}
```

7.8 Component Reference

- **pan-data-connector**: REST API integration - See Chapter 20
- **pan-graphql-connector**: GraphQL integration - See Chapter 20
- **pan-websocket**: WebSocket connection management - See Chapter 20
- **pan-idb**: IndexedDB caching - See Chapter 18

7.9 Complete Example: Product Search with Caching

```
class ProductSearch extends LarcComponent {
  constructor() {
    super();
    this.products = [];
    this.loading = false;
    this.error = null;
    this.searchTerm = '';
    this.debounceTimer = null;
  }

  async connectedCallback() {
    // Load cached results
    const cached = this.getCache('products-all');
    if (cached) {
      this.products = cached;
      this.render();
    }

    // Fetch fresh data
    await this.loadProducts();
  }

  async loadProducts(search = '') {
    this.loading = true;
    this.error = null;
    this.render();

    try {
      const url = search
        ? `/api/products/search?q=${encodeURIComponent(search)}`
        : '/api/products';
```

```
      const response = await fetch(url);
      if (!response.ok) throw new Error('Search failed');

      this.products = await response.json();

      // Cache full product list only
      if (!search) {
        this.setCache('products-all', this.products, 5 * 60 * 1000);
      }
    } catch (err) {
      this.error = err.message;
    } finally {
      this.loading = false;
      this.render();
    }
  }

  handleSearchInput(event) {
    this.searchTerm = event.target.value;

    // Debounce search
    clearTimeout(this.debounceTimer);
    this.debounceTimer = setTimeout(() => {
      this.loadProducts(this.searchTerm);
    }, 300);
  }

  getCache(key) {
    const item = localStorage.getItem(`cache:${key}`);
    if (!item) return null;

    const { data, expires } = JSON.parse(item);
    return Date.now() < expires ? data : null;
```

```
  }

  setCache(key, data, ttl) {
    localStorage.setItem(`cache:${key}`, JSON.stringify({
      data,
      expires: Date.now() + ttl
    }));
  }

  template() {
    return `
      <div class="product-search">
        <input
          type="search"
          placeholder="Search products..."
          value="${this.searchTerm}"
          oninput="this.handleSearchInput(event)">

        ${this.loading ? '<div class="spinner">Searching...</div>'
        : ''}
        ${this.error ? `<div class="error">${this.error}</div>` :
        ''}

        <div class="product-grid">
          ${this.products.map(product => `
            <div class="product-card">
              <img src="${product.image}" alt="${product.name}">
              <h3>${product.name}</h3>
              <p class="price">$${product.price}</p>
              <button onclick="addToCart(${product.id})">Add to
      Cart</button>
            </div>
          `).join('')}
        </div>
```

```
        ${this.products.length === 0 && !this.loading ? '
          <div class="no-results">No products found</div>
        ' : ''}
      </div>
    `;
  }
}

customElements.define('product-search', ProductSearch);
```

7.10 Cross-References

- **Tutorial**: *Learning LARC* Chapter 11 (Data Fetching and APIs)
- **Components**: Chapter 20 (pan-data-connector, pan-graphql-connector, pan-websocket)
- **Patterns**: Appendix E (API Integration Patterns)
- **Related**: Chapter 4 (State Management), Chapter 9 (Realtime Features)

7.11 Common Issues

7.11.1 Issue: CORS errors in development

Problem: `Access-Control-Allow-Origin` errors **Solution**: Configure dev server proxy or add CORS headers to API

7.11.2 Issue: Stale cache data

Problem: Users see outdated information **Solution**: Implement cache invalidation with

TTL, version keys, or manual clear

7.11.3 Issue: Request waterfall

Problem: Sequential requests slow down page load **Solution**: Batch requests, use GraphQL, or fetch data in parallel

7.11.4 Issue: Memory leaks from pending requests

Problem: Component unmounts but fetch continues **Solution**: Use AbortController to cancel requests in `disconnectedCallback()`

7.11.5 Issue: Authentication tokens expired

Problem: 401 errors on API calls **Solution**: Implement token refresh interceptor before retrying failed requests

See *Learning LARC* Chapter 11 for detailed API patterns, authentication flows, and advanced caching strategies.

8 Authentication and Authorization

Quick reference for authentication and authorization patterns in LARC applications. For detailed tutorials, see *Learning LARC* Chapter 12.

8.1 Overview

Authentication verifies user identity ("who are you?") while authorization determines permissions ("what can you do?"). LARC applications typically use JWT tokens for authentication and role-based access control (RBAC) for authorization.

Key Concepts:

- JWT tokens: Cryptographically signed identity tokens with claims
- Token refresh: Automatic renewal before expiry
- Protected routes: Enforce authentication/authorization requirements
- RBAC: Role-based permission system with inheritance
- Session management: Timeout, activity tracking, secure storage

8.2 Quick Example

```
// Initialize authentication
import { authService } from './services/auth.js';

await authService.initialize();

if (authService.getState().isAuthenticated) {
  // User logged in
  console.log('User:', authService.getState().user);
} else {
  // Redirect to login
  window.location.href = '/login';
}

// Login
const success = await authService.login({
  username: 'alice',
  password: 'secret123'
});

// Check permissions
if (authService.hasRole('admin')) {
  // Show admin features
}
```

8.3 Authentication Service API

Method	Parameters	Returns	Description
`initialize()`	-	Promise<boolean>	Load stored tokens, verify, auto-refresh if needed
`login(credentials)`	{username, password}	Promise<boolean>	Authenticate and store tokens
`logout()`	-	void	Clear tokens and state
`getState()`	-	AuthState	Get current authentication state
`getAccessToken()`	-	string|null	Get token for API requests
`hasRole(role)`	role: string	boolean	Check if user has role
`hasPermission(perm)`	permission: string	boolean	Check if user has permission

8.3.1 AuthState Interface

```
interface AuthState {
  isAuthenticated: boolean;
  user: UserClaims | null;
  tokens: AuthTokens | null;
}

interface UserClaims {
  userId: string;
  username: string;
  roles: string[];
  permissions: string[];
}

interface AuthTokens {
  accessToken: string;
  refreshToken: string;
  expiresIn: number;
}
```

8.4 API Interceptor Pattern

Automatically add authentication headers to requests:

```
// Add bearer token to requests
api.interceptors.request.use(async (config) => {
  const token = authService.getAccessToken();
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

// Handle 401 by refreshing token
api.interceptors.response.use(
  (response) => response,
  async (error) => {
    if (error.response?.status === 401 && !error.config._retry) {
      error.config._retry = true;
      await authService.refreshAccessToken();

      const newToken = authService.getAccessToken();
      error.config.headers.Authorization = `Bearer ${newToken}`;
      return api(error.config);
    }
    return Promise.reject(error);
  }
);
```

8.5 Protected Routes

8.5.1 Route Guard Component

```
class ProtectedRoute extends HTMLElement {
  connectedCallback() {
    const state = authService.getState();

    if (!state.isAuthenticated) {
      const redirect = encodeURIComponent(window.location.pathname);
      window.location.href = `/login?redirect=${redirect}`;
      return;
    }

    // Check roles
    const requiredRoles = this.getAttribute('roles')?.split(',') ||
        [];
    if (requiredRoles.length) {
      const hasRole = requiredRoles.some(r =>
        authService.hasRole(r));
      if (!hasRole) {
        window.location.href = '/forbidden';
        return;
      }
    }
  }
}

customElements.define('protected-route', ProtectedRoute);
```

8.5.2 Usage

```
<!-- Public route -->
<route path="/" component="home-page"></route>

<!-- Requires authentication -->
<route path="/dashboard">
  <protected-route>
    <dashboard-page></dashboard-page>
  </protected-route>
</route>

<!-- Requires admin role -->
<route path="/admin">
  <protected-route roles="admin">
    <admin-panel></admin-panel>
  </protected-route>
</route>
```

8.6 Role-Based Access Control

8.6.1 RBAC Configuration

```
const ROLES = {
  guest: {
    id: 'guest',
    permissions: [
      { resource: 'content', action: 'read', scope: 'public' }
    ]
  },

  user: {
    id: 'user',
    inherits: ['guest'],
    permissions: [
      { resource: 'profile', action: 'read', scope: 'own' },
      { resource: 'profile', action: 'update', scope: 'own' },
      { resource: 'content', action: 'create', scope: 'own' }
    ]
  },

  moderator: {
    id: 'moderator',
    inherits: ['user'],
    permissions: [
      { resource: 'content', action: 'update', scope: 'all' },
      { resource: 'content', action: 'delete', scope: 'all' }
    ]
  },

  admin: {
    id: 'admin',
    inherits: ['moderator'],
    permissions: [
      { resource: 'users', action: 'create' },
```

```
      { resource: 'users', action: 'update' },
      { resource: 'users', action: 'delete' },
      { resource: 'settings', action: 'update' }
    ]
  }
};
```

8.6.2 Authorization Service API

Method	Parameters	Returns	Description
`getPermissions(roleId)`	roleId: string	Permission[]	Get all permissions (including inherited)
`hasPermission(roles, resource, action, scope?)`	roles: string[], resource: string, action: string, scope?: string	boolean	Check if any role has permission
`canAccess(roles, resource, action, ownerId?, userId?)`	roles: string[], resource: string, action: string, ownerId?: string, userId?: string	boolean	Check access with ownership

8.6.3 Conditional Rendering

```
class AuthorizedContent extends HTMLElement {
  connectedCallback() {
    const resource = this.getAttribute('resource');
    const action = this.getAttribute('action');

    const state = authService.getState();
    const authorized = state.user &&
      authz.hasPermission(state.user.roles, resource, action);

    if (!authorized) {
      this.style.display = 'none';
    }
  }
}

customElements.define('authorized-content', AuthorizedContent);
```

8.7 Session Management

8.7.1 Session Timeout

```
class SessionManager {
  constructor(timeoutMinutes, warningMinutes) {
    this.timeoutMinutes = timeoutMinutes;
    this.warningMinutes = warningMinutes;
    this.setupActivityListeners();
  }

  start() {
    this.resetTimeout();
  }

  resetTimeout() {
    clearTimeout(this.timeoutId);
    clearTimeout(this.warningId);

    // Warning before timeout
    const warningMs = this.warningMinutes * 60 * 1000;
    this.warningId = setTimeout(() => {
      this.showWarning();
    }, warningMs);

    // Logout on timeout
    const timeoutMs = this.timeoutMinutes * 60 * 1000;
    this.timeoutId = setTimeout(() => {
      authService.logout();
      window.location.href = '/login?reason=timeout';
    }, timeoutMs);
  }

  setupActivityListeners() {
    ['mousedown', 'keydown', 'scroll', 'touchstart'].forEach(event
        => {
```

```
      document.addEventListener(event, () => {
        if (authService.getState().isAuthenticated) {
          this.resetTimeout();
        }
      }, { passive: true });
    });
  }
}

// Initialize with 30-minute timeout
const sessionManager = new SessionManager(30, 25);
sessionManager.start();
```

8.8 Security Best Practices

8.8.1 Input Validation

```
const validators = {
  email: (value) => /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(value),

  password: (value) => {
    const errors = [];
    if (value.length < 8) errors.push('8+ characters required');
    if (!/[A-Z]/.test(value)) errors.push('Uppercase required');
    if (!/[a-z]/.test(value)) errors.push('Lowercase required');
    if (!/[0-9]/.test(value)) errors.push('Number required');
    if (!/[^A-Za-z0-9]/.test(value)) errors.push('Special char
        required');
    return { valid: errors.length === 0, errors };
  },

  sanitize: (value) =>
    value.replace(/[<>]/g, '')
         .replace(/javascript:/gi, '')
         .trim()
};
```

8.8.2 CSRF Protection

```
class CSRFProtection {
  generateToken() {
    const token = this.randomString(32);
    sessionStorage.setItem('csrf-token', token);
    return token;
  }

  getToken() {
    return sessionStorage.getItem('csrf-token') || '';
  }

  addToHeaders(headers) {
    return {
      ...headers,
      'X-CSRF-Token': this.getToken()
    };
  }

  randomString(length) {
    const chars =
        'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';

    const array = new Uint8Array(length);
    crypto.getRandomValues(array);
    return Array.from(array).map(x => chars[x %
        chars.length]).join('');
  }
}

const csrf = new CSRFProtection();
```

8.9 Component Reference

See Chapter 20 for authentication-related components:

- **pan-auth**: Complete authentication component
- **pan-user-menu**: User profile dropdown with logout

8.10 Complete Example: Login Form

```
class LoginForm extends HTMLElement {
  connectedCallback() {
    this.state = {
      username: '',
      password: '',
      loading: false,
      error: null
    };

    csrf.generateToken();
    this.render();
  }

  async handleSubmit(e) {
    e.preventDefault();

    this.state.loading = true;
    this.state.error = null;
    this.render();

    try {
      const success = await authService.login({
        username: this.state.username,
        password: this.state.password
      });

      if (success) {
        const params = new URLSearchParams(window.location.search);
        const redirect = params.get('redirect') || '/dashboard';
        window.location.href = redirect;
      } else {
        this.state.error = 'Invalid credentials';
```

```
      }
    } catch (err) {
      this.state.error = 'Login failed. Try again.';
    } finally {
      this.state.loading = false;
      this.render();
    }
  }

  render() {
    this.innerHTML = `
      <form class="login-form">
        ${this.state.error ? `<div
        class="error">${this.state.error}</div>` : ''}

        <label>
          Username
          <input name="username" value="${this.state.username}"
        required>
        </label>

        <label>
          Password
          <input type="password" name="password"
        value="${this.state.password}" required>
        </label>

        <button type="submit" ${this.state.loading ? 'disabled' :
        ''}>
          ${this.state.loading ? 'Signing in...' : 'Sign In'}
        </button>
      </form>
    `;
```

```
    this.querySelector('form').addEventListener('submit', (e) =>
        this.handleSubmit(e));
    this.querySelector('[name="username"]').addEventListener('input', (e)
        => {

      this.state.username = e.target.value;
    });
    this.querySelector('[name="password"]').addEventListener('input', (e)
        => {

      this.state.password = e.target.value;
    });
  }
}

customElements.define('login-form', LoginForm);
```

8.11 Cross-References

- **Tutorial**: *Learning LARC* Chapter 12 (Authentication and Authorization)
- **Components**: Chapter 20 (pan-auth, pan-user-menu)
- **Patterns**: Appendix E (Security Patterns)
- **Related**: Chapter 7 (API Authentication), Chapter 9 (WebSocket Authentication)

8.12 Common Issues

8.12.1 Issue: Token expires during request

Problem: 401 errors on long-running requests **Solution**: Refresh token before expiry (subtract 5 minutes from expiry time)

8.12.2 Issue: Lost authentication on page refresh

Problem: User logged out after reload **Solution**: Call `authService.initialize()` on app startup to restore session

8.12.3 Issue: Infinite redirect loops

Problem: Protected route redirects to login, login redirects to protected route **Solution**: Check authentication before redirect; use redirect parameter correctly

8.12.4 Issue: CORS errors with credentials

Problem: `Access-Control-Allow-Credentials` errors **Solution**: Set `credentials: 'include'` in fetch options, enable CORS on server

8.12.5 Issue: XSS attacks via stored tokens

Problem: Token accessible via JavaScript **Solution**: Use httpOnly cookies for refresh tokens; store access tokens in memory when possible

See *Learning LARC* Chapter 12 for complete authentication flows, OAuth integration, and multi-factor authentication patterns.

9 Real-time Features

Quick reference for real-time communication patterns in LARC applications. For detailed tutorials, see *Learning LARC* Chapter 11.

9.1 Overview

Real-time features enable live data updates without page reloads using WebSockets, Server-Sent Events (SSE), BroadcastChannel for cross-tab sync, and Web Workers for background processing.

Key Concepts:

- WebSockets: Full-duplex bidirectional communication
- SSE: One-way server-to-client streaming
- BroadcastChannel: Cross-tab/window messaging
- Web Workers: Background thread processing
- Heartbeat: Keep-alive mechanism to detect disconnections
- Reconnection: Automatic recovery from connection failures

9.2 Quick Example

```
// WebSocket connection
import { wsClient } from './services/websocket-client.js';

await wsClient.connect();

// Subscribe to events
wsClient.on('notification', (data) => {
  console.log('New notification:', data);
});

// Send message
wsClient.send('chat.message', { text: 'Hello!' });

// Check connection
if (wsClient.isConnected()) {
  // Connected
}
```

9.3 WebSocket Client API

Method	Parameters	Returns	Description
`connect()`	-	Promise<void>	Connect to WebSocket server
`disconnect()`	-	void	Close connection
`send(type, payload)`	type: string, payload: any	void	Send message to server
`on(type, handler)`	type: string, handler: Function	Function	Subscribe to message type (returns unsubscribe)
`isConnected()`	-	boolean	Check connection status

9.3.1 WebSocket Configuration

```
class WebSocketClient {
  constructor(config) {
    this.config = {
      url: 'ws://localhost:3000/ws',
      reconnectInterval: 5000,        // Delay between reconnect
        attempts
      maxReconnectAttempts: 10,       // Max reconnection attempts
      heartbeatInterval: 30000,       // Heartbeat ping interval
      ...config
    };
  }
}
```

9.3.2 WebSocket Connection Pattern

```
const wsClient = new WebSocketClient({
  url: 'ws://localhost:3000/ws'
});

// Connection lifecycle
wsClient.on('connected', () => {
  console.log('Connected');
});

wsClient.on('disconnected', ({ code, reason }) => {
  console.log('Disconnected:', code, reason);
});

wsClient.on('error', ({ error }) => {
  console.error('WebSocket error:', error);
});

wsClient.on('reconnect-failed', () => {
  console.error('Reconnection failed');
});

await wsClient.connect();
```

9.4 Server-Sent Events (SSE)

SSE provides one-way server-to-client streaming over HTTP. Simpler than WebSockets, automatic reconnection, works with existing HTTP infrastructure.

9.4.1 SSE Client API

Method	Parameters	Returns	Description
`connect()`	-	void	Connect to SSE endpoint
`disconnect()`	-	void	Close connection
`on(eventName, handler)`	eventName: string, handler: Function	Function	Subscribe to named events
`isConnected()`	-	boolean	Check connection status

9.4.2 SSE Usage Pattern

```
import { SSEClient } from './services/sse-client.js';

const sseClient = new SSEClient({
  url: '/api/events',
  withCredentials: true,
  reconnectDelay: 3000
});

sseClient.connect();

// Subscribe to named events
sseClient.on('activity', (data) => {
  console.log('Activity:', data);
});

sseClient.on('notification', (data) => {
  console.log('Notification:', data);
});

// Connection events
sseClient.on('connected', () => console.log('SSE connected'));
sseClient.on('disconnected', () => console.log('SSE disconnected'));
```

9.5 BroadcastChannel: Cross-Tab Communication

Synchronize state across browser tabs and windows.

9.5.1 BroadcastChannel API

```
class TabSyncService {
  constructor(channelName = 'app-sync') {
    this.channel = new BroadcastChannel(channelName);
    this.tabId = this.generateTabId();

    this.channel.onmessage = (event) => {
      this.handleMessage(event.data);
    };
  }

  broadcast(type, payload) {
    this.channel.postMessage({
      type,
      payload,
      timestamp: Date.now(),
      tabId: this.tabId
    });
  }

  on(type, handler) {
    // Subscribe to message type
  }
}

const tabSync = new TabSyncService();
```

9.5.2 Cross-Tab Sync Patterns

```
// Sync authentication state
tabSync.on('auth-logout', () => {
  authService.logout();
  window.location.href = '/login';
});

tabSync.on('auth-login', (user) => {
  window.location.reload();
});

// Broadcast logout to other tabs
tabSync.broadcast('auth-logout', {});

// Sync data changes
tabSync.on('data-updated', ({ resource, id }) => {
  // Refresh data in this tab
  reloadResource(resource, id);
});
```

9.6 Web Workers: Background Processing

Run JavaScript in background threads without blocking UI.

9.6.1 Worker Creation

```
// worker.js
self.onmessage = (event) => {
  const { id, type, data, options } = event.data;

  try {
    let result;

    switch (type) {
      case 'sort':
        result = sortData(data, options);
        break;
      case 'filter':
        result = filterData(data, options);
        break;
      case 'aggregate':
        result = aggregateData(data, options);
        break;
    }

    self.postMessage({ id, result });
  } catch (error) {
    self.postMessage({ id, result: null, error: error.message });
  }
};

function sortData(data, options) {
  const { field, order = 'asc' } = options;
  return [...data].sort((a, b) => {
    const comparison = a[field] < b[field] ? -1 : a[field] > b[field] ?
        1 : 0;
    return order === 'asc' ? comparison : -comparison;
  });
```

```
}
```

9.6.2 Worker Manager

```
class WorkerManager {
  constructor(workerUrl) {
    this.worker = new Worker(workerUrl);
    this.requestId = 0;
    this.pendingRequests = new Map();

    this.worker.onmessage = (event) => {
      const { id, result, error } = event.data;
      const pending = this.pendingRequests.get(id);

      if (pending) {
        this.pendingRequests.delete(id);
        error ? pending.reject(new Error(error)) :
        pending.resolve(result);
      }
    };
  }

  async process(type, data, options) {
    const id = `req-${++this.requestId}`;

    return new Promise((resolve, reject) => {
      this.pendingRequests.set(id, { resolve, reject });
      this.worker.postMessage({ id, type, data, options });

      setTimeout(() => {
        if (this.pendingRequests.has(id)) {
          this.pendingRequests.delete(id);
          reject(new Error('Request timeout'));
        }
      }, 30000);
    });
```

```
  }

  terminate() {
    this.worker.terminate();
    this.pendingRequests.forEach(({ reject }) => {
      reject(new Error('Worker terminated'));
    });
    this.pendingRequests.clear();
  }
}

const dataWorker = new WorkerManager('/workers/data-processor.js');

// Use worker
const sorted = await dataWorker.process('sort', data, {
  field: 'name',
  order: 'asc'
});
```

9.7 Real-time Component Patterns

9.7.1 Notification Feed

```
class NotificationFeed extends HTMLElement {
  connectedCallback() {
    this.notifications = [];

    wsClient.on('notification', (notification) => {
      this.notifications.unshift(notification);
      this.render();

      // Auto-dismiss after 5s
      setTimeout(() => {
        this.removeNotification(notification.id);
      }, 5000);
    });

    this.render();
  }

  render() {
    this.innerHTML = `
      <div class="notifications">
        ${this.notifications.map(n => `
          <div class="notification ${n.type}">
            <strong>${n.title}</strong>
            <p>${n.message}</p>
          </div>
        `).join('')}
      </div>
    `;
  }
}

customElements.define('notification-feed', NotificationFeed);
```

9.7.2 Live Activity Feed

```
class ActivityFeed extends HTMLElement {
  async connectedCallback() {
    this.activities = [];

    // Load initial data
    const response = await fetch('/api/activities');
    this.activities = await response.json();

    // Subscribe to live updates
    sseClient.on('activity', (activity) => {
      this.activities.unshift(activity);
      this.activities = this.activities.slice(0, 100); // Limit
      this.render();
    });

    this.render();
  }

  render() {
    this.innerHTML = `
      <div class="activities">
        ${this.activities.map(a => `
          <div class="activity">
            <strong>${a.userName}</strong> ${a.action}
        <em>${a.target}</em>
            <span
        class="time">${this.formatTime(a.timestamp)}</span>
          </div>
        `).join('')}
      </div>
    `;
  }
```

```
  formatTime(timestamp) {
    const diff = Date.now() - timestamp;
    if (diff < 60000) return 'Just now';
    if (diff < 3600000) return `${Math.floor(diff / 60000)}m ago`;
    return new Date(timestamp).toLocaleString();
  }
}

customElements.define('activity-feed', ActivityFeed);
```

9.7.3 Collaborative Editor

```
class CollaborativeEditor extends HTMLElement {
  connectedCallback() {
    this.docId = this.getAttribute('doc-id');
    this.content = '';
    this.collaborators = new Map();

    // Join document
    wsClient.send('join-document', { docId: this.docId });

    // Subscribe to updates
    wsClient.on('document-update', (update) => {
      if (!this.isLocalUpdate) {
        this.content = update.content;
        this.render();
      }
    });

    wsClient.on('collaborator-joined', (collab) => {
      this.collaborators.set(collab.userId, collab);
      this.render();
    });

    wsClient.on('collaborator-left', ({ userId }) => {
      this.collaborators.delete(userId);
      this.render();
    });

    this.render();
  }

  handleInput(e) {
    this.content = e.target.value;
```

```
    this.isLocalUpdate = true;

    wsClient.send('document-update', {
      docId: this.docId,
      content: this.content
    });

    setTimeout(() => { this.isLocalUpdate = false; }, 100);
  }

  render() {
    this.innerHTML = `
      <div class="editor">
        <div class="collaborators">
          ${Array.from(this.collaborators.values()).map(c => `
            <div class="badge">${c.name[0]}</div>
          `).join('')}
        </div>
        <textarea>${this.content}</textarea>
      </div>
    `;

    this.querySelector('textarea').addEventListener('input', (e) =>
        this.handleInput(e));
  }
}

customElements.define('collaborative-editor', CollaborativeEditor);
```

9.8 Component Reference

See Chapter 20 for real-time components:

- **pan-websocket**: WebSocket connection management
- **pan-sse**: Server-Sent Events integration
- **pan-live-data**: Auto-refreshing data display

9.9 Advanced Patterns

9.9.1 Optimistic Updates

```
class OptimisticList extends HTMLElement {
  async addItem(item) {
    // Add to UI immediately (optimistic)
    this.items.push({ ...item, pending: true });
    this.render();

    try {
      // Send to server
      const response = await fetch('/api/items', {
        method: 'POST',
        body: JSON.stringify(item)
      });

      const serverItem = await response.json();

      // Replace pending with server version
      const index = this.items.findIndex(i => i.pending);
      this.items[index] = serverItem;
      this.render();
    } catch (err) {
      // Rollback on error
      this.items = this.items.filter(i => !i.pending);
      this.render();
      alert('Failed to add item');
    }
  }
}
```

9.9.2 Presence Tracking

```
class PresenceTracker {
  constructor() {
    this.users = new Map();

    wsClient.on('presence-update', ({ userId, status }) => {
      this.users.set(userId, { userId, status, lastSeen: Date.now()
        });
      this.notifySubscribers();
    });

    wsClient.on('presence-offline', ({ userId }) => {
      this.users.delete(userId);
      this.notifySubscribers();
    });

    // Send heartbeat
    setInterval(() => {
      wsClient.send('presence-heartbeat', {});
    }, 30000);
  }

  getOnlineUsers() {
    return Array.from(this.users.values()).filter(u => u.status ===
        'online');
  }
}
```

9.9.3 Conflict Resolution

```
class ConflictResolver {
  async handleUpdate(localVersion, remoteVersion) {
    if (localVersion.timestamp > remoteVersion.timestamp) {
      // Local wins - send to server
      await this.sendUpdate(localVersion);
    } else if (localVersion.timestamp < remoteVersion.timestamp) {
      // Remote wins - apply locally
      this.applyUpdate(remoteVersion);
    } else {
      // Timestamps equal - merge
      const merged = this.merge(localVersion, remoteVersion);
      this.applyUpdate(merged);
    }
  }

  merge(local, remote) {
    // Last-write-wins per field
    return {
      ...remote,
      ...Object.entries(local).reduce((acc, [key, value]) => {
        if (local[`${key}_timestamp`] > remote[`${key}_timestamp`])
        {
          acc[key] = value;
        }
        return acc;
      }, {})
    };
  }
}
```

9.10 Performance Considerations

Strategy	Use Case	Implementation
Throttling	High-frequency events (scroll, mousemove)	Send update max once per 100-500ms
Debouncing	Text input	Send update after 300ms of inactivity
Batching	Multiple updates	Accumulate and send in single message
Compression	Large payloads	Use MessagePack or gzip compression
Selective sync	Large datasets	Only sync visible/relevant data

9.10.1 Throttle Example

```
class ThrottledInput extends HTMLElement {
  connectedCallback() {
    let lastSent = 0;
    const throttleMs = 300;

    this.querySelector('input').addEventListener('input', (e) => {
      const now = Date.now();
      if (now - lastSent >= throttleMs) {
        wsClient.send('input-update', { value: e.target.value });
        lastSent = now;
      }
    });
  }
}
```

9.11 Cross-References

- **Tutorial**: *Learning LARC* Chapter 11 (Real-time Features)
- **Components**: Chapter 20 (pan-websocket, pan-sse, pan-live-data)
- **Patterns**: Appendix E (Real-time Patterns)
- **Related**: Chapter 7 (API Integration), Chapter 12 (Performance)

9.12 Common Issues

9.12.1 Issue: Connection drops on mobile/sleep

Problem: WebSocket closes when device sleeps **Solution**: Implement heartbeat + reconnection; use SSE for mobile

9.12.2 Issue: Message order not guaranteed

Problem: Messages arrive out of sequence **Solution**: Add sequence numbers; buffer and reorder on client

9.12.3 Issue: Memory leaks from event listeners

Problem: Component unmounts but listeners remain **Solution**: Always unsubscribe in `disconnectedCallback()`

9.12.4 Issue: Duplicate messages on reconnect

Problem: Server resends messages after reconnection **Solution**: Track message IDs; deduplicate on client

9.12.5 Issue: High bandwidth usage

Problem: Too many messages or large payloads **Solution**: Throttle updates; compress payloads; batch messages

See *Learning LARC* Chapter 11 for complete real-time patterns, WebSocket authentication, and scaling strategies.

10 File Management

Quick reference for file management and OPFS (Origin Private File System) in LARC applications. For detailed tutorials, see *Learning LARC* Chapter 11.

10.1 Overview

OPFS provides browser-based file storage with high performance, large capacity (GB+), and persistence across sessions. Unlike localStorage (5-10MB) or IndexedDB, OPFS offers native file system operations ideal for offline-first apps and large file handling.

Key Concepts:

- OPFS: Origin Private File System - browser's private file storage
- FileSystemDirectoryHandle: Directory reference for operations
- FileSystemFileHandle: File reference for read/write operations
- Storage quota: Browser-managed space limits (best-effort persistence)
- Streaming: Handle large files without loading into memory

10.2 Quick Example

```
// Initialize OPFS
const root = await navigator.storage.getDirectory();

// Write file
const fileHandle = await root.getFileHandle('notes.txt', { create:
        true });
const writable = await fileHandle.createWritable();
await writable.write('Hello OPFS!');
await writable.close();

// Read file
const file = await fileHandle.getFile();
const text = await file.text();
console.log(text); // "Hello OPFS!"

// List files
for await (const [name, handle] of root.entries()) {
  console.log(name, handle.kind); // 'notes.txt' 'file'
}
```

10.3 FileSystemService API

Method	Parameters	Returns	Descriptio
`initialize()`	-	Promise<void>	Initialize OPFS root handle
`createDirectory(path)`	path: string	Promise<DirectoryHandle>	Create directory (recursive)
`getDirectory(path)`	path: string	Promise<DirectoryHandle>	Get existin directory
`listFiles(path)`	path?: string	Promise<FileInfo[]>	List files in directory
`listDirectories(path)`	path?: string	Promise<DirectoryInfo[]>	List subdirecto
`writeFile(path, name, content)`	path: string, name: string, content: Blob|ArrayBuffer|string	Promise<void>	Write or update file
`readFile(path, name)`	path: string, name: string	Promise<File>	Read file contents
`deleteFile(path, name)`	path: string, name: string	Promise<void>	Delete file
`deleteDirectory(path, recursive?)`	path: string, recursive?: boolean	Promise<void>	Delete directory

Method	Parameters	Returns	Descriptio
`fileExists(path, name)`	path: string, name: string	Promise<boolean>	Check if fil exists
`copyFile(src, dest)`	sourcePath: string, sourceFile: string, destPath: string, destFile: string	Promise<void>	Copy file
`moveFile(src, dest)`	sourcePath: string, sourceFile: string, destPath: string, destFile: string	Promise<void>	Move file
`getStorageInfo()`	-	Promise<StorageInfo>	Get quota and usage stats
`getDirectorySize(path)`	path?: string	Promise<number>	Calculate total size (bytes)

10.3.1 FileInfo Interface

```
interface FileInfo {
  name: string;
  size: number;
  type: string;
  handle: FileSystemFileHandle;
  lastModified: number;
}
```

10.4 Common Operations

10.4.1 Create and Write File

```
const fileSystem = new FileSystemService();
await fileSystem.initialize();

// Write text
await fileSystem.writeFile('/', 'note.txt', 'Hello World');

// Write binary
const blob = new Blob(['data'], { type: 'application/octet-stream'
        });
await fileSystem.writeFile('/documents', 'file.bin', blob);

// Write ArrayBuffer
const buffer = new ArrayBuffer(1024);
await fileSystem.writeFile('/data', 'binary.dat', buffer);
```

10.4.2 Read and Process Files

```
// Read file
const file = await fileSystem.readFile('/', 'note.txt');
const text = await file.text();

// Stream large files
const file = await fileSystem.readFile('/', 'large-file.mp4');
const stream = file.stream();
const reader = stream.getReader();

while (true) {
  const { done, value } = await reader.read();
  if (done) break;
  processChunk(value);
}
```

10.4.3 Directory Operations

```
// Create nested directories
await fileSystem.createDirectory('/documents/2024/reports');

// List with filtering
const files = await fileSystem.listFiles('/documents');
const images = files.filter(f => f.type.startsWith('image/'));

// Recursive size calculation
const size = await fileSystem.getDirectorySize('/documents');
console.log(`Total: ${size} bytes`);
```

10.5 Storage Quota Management

10.5.1 Check Quota

```
const { usage, quota, percentUsed } = await
        fileSystem.getStorageInfo();

console.log(`Used: ${usage} / ${quota} bytes (${percentUsed}%)`);

if (percentUsed > 75) {
  alert('Running low on storage');
}
```

10.5.2 Request Persistent Storage

```
// Request persistence (prevents automatic eviction)
if (navigator.storage?.persist) {
  const isPersisted = await navigator.storage.persist();
  console.log('Persistent:', isPersisted);
}

// Check current persistence status
if (navigator.storage?.persisted) {
  const persisted = await navigator.storage.persisted();
  console.log('Currently persisted:', persisted);
}
```

10.6 File Upload Pattern

10.6.1 Drag-and-Drop Upload

```
class FileUploadComponent extends HTMLElement {
  connectedCallback() {
    this.addEventListener('drop', async (e) => {
      e.preventDefault();

      const files = Array.from(e.dataTransfer.files);
      for (const file of files) {
        await this.uploadFile(file);
      }
    });

    this.addEventListener('dragover', (e) => e.preventDefault());
  }

  async uploadFile(file) {
    const content = await file.arrayBuffer();
    await fileSystem.writeFile('/', file.name, content);
    this.dispatchEvent(new CustomEvent('file-uploaded', {
      detail: { name: file.name, size: file.size }
    }));
  }
}
```

10.6.2 Upload with Progress Tracking

```
async function uploadWithProgress(file, onProgress) {
  const chunkSize = 1024 * 1024; // 1MB chunks
  const chunks = Math.ceil(file.size / chunkSize);

  const fileHandle = await root.getFileHandle(file.name, { create:
        true });
  const writable = await fileHandle.createWritable();

  for (let i = 0; i < chunks; i++) {
    const start = i * chunkSize;
    const end = Math.min(start + chunkSize, file.size);
    const chunk = file.slice(start, end);

    await writable.write(chunk);
    onProgress((i + 1) / chunks * 100);
  }

  await writable.close();
}

// Usage
await uploadWithProgress(file, (percent) => {
  console.log(`Upload: ${percent}%`);
});
```

10.7 File Download Pattern

```
async function downloadFile(path, fileName) {
  const file = await fileSystem.readFile(path, fileName);
  const url = URL.createObjectURL(file);

  const a = document.createElement('a');
  a.href = url;
  a.download = fileName;
  a.click();

  URL.revokeObjectURL(url);
}
```

10.8 File Browser Component

```
class FileBrowser extends HTMLElement {
  async connectedCallback() {
    this.currentPath = '/';
    await this.render();
  }

  async render() {
    const files = await fileSystem.listFiles(this.currentPath);
    const dirs = await fileSystem.listDirectories(this.currentPath);

    this.innerHTML = `
      <div class="breadcrumb">${this.currentPath}</div>

      ${dirs.map(d => `
        <div class="dir" onclick="navigate('${d.name}')">
          📁 ${d.name}
        </div>
      `).join('')}

      ${files.map(f => `
        <div class="file" onclick="open('${f.name}')">
          ${this.getIcon(f.type)} ${f.name}
          (${this.formatSize(f.size)})
        </div>
      `).join('')}
    `;
  }

  getIcon(type) {
    if (type.startsWith('image/')) return '🖼️';
    if (type.startsWith('video/')) return '🎬';
    if (type.startsWith('audio/')) return '🎵';
```

```
    if (type === 'application/pdf') return '📕';
    return '📄';
  }

  formatSize(bytes) {
    const units = ['B', 'KB', 'MB', 'GB'];
    let size = bytes;
    let unitIndex = 0;

    while (size >= 1024 && unitIndex < units.length - 1) {
      size /= 1024;
      unitIndex++;
    }

    return `${size.toFixed(1)} ${units[unitIndex]}`;
  }
}

customElements.define('file-browser', FileBrowser);
```

10.9 Search and Filter

```
class FileSearch {
  async search(query, options = {}) {
    const results = [];
    await this.searchDirectory('/', query, options, results);
    return results.sort((a, b) => b.score - a.score);
  }

  async searchDirectory(path, query, options, results) {
    const files = await fileSystem.listFiles(path);

    for (const file of files) {
      const score = this.matchFile(file, query, options);
      if (score > 0) {
        results.push({ file, path, score });
      }
    }

    if (options.recursive) {
      const dirs = await fileSystem.listDirectories(path);
      for (const dir of dirs) {
        const subPath = `${path}/${dir.name}`.replace('//', '/');
        await this.searchDirectory(subPath, query, options,
         results);
      }
    }
  }

  matchFile(file, query, options) {
    const name = file.name.toLowerCase();
    const q = query.toLowerCase();

    let score = 0;
```

```
    if (name === q) score += 100;
    else if (name.startsWith(q)) score += 50;
    else if (name.includes(q)) score += 25;
    else return 0;

    // Filter by type
    if (options.types?.length) {
      const matches = options.types.some(t =>
        file.type.startsWith(t));
      if (!matches) return 0;
      score += 10;
    }

    // Filter by size
    if (options.minSize && file.size < options.minSize) return 0;
    if (options.maxSize && file.size > options.maxSize) return 0;

    return score;
  }
}

const fileSearch = new FileSearch();

// Usage
const results = await fileSearch.search('photo', {
  types: ['image/'],
  minSize: 1024 * 100, // 100KB min
  recursive: true
});
```

10.10 Component Reference

See Chapter 19 for file management UI components:

- **pan-files**: Complete file manager with browser, upload, quota display
- **pan-file-picker**: File selection dialog
- **pan-image-viewer**: Image preview and editing

10.11 Performance Tips

Optimization	Implementation	Benefit
Stream large files	Use `file.stream()` instead of `file.arrayBuffer()`	Reduce memory usage
Batch operations	Group multiple writes in single transaction	Improve write speed
Cache handles	Store FileSystemHandle references	Reduce lookup overhead
Lazy loading	Load directory contents on demand	Faster initial render
Background cleanup	Delete temp files in Web Worker	Non-blocking UI

10.11.1 Streaming Example

```
async function processLargeFile(path, fileName) {
  const file = await fileSystem.readFile(path, fileName);
  const stream = file.stream();
  const reader = stream.getReader();

  let bytesProcessed = 0;

  while (true) {
    const { done, value } = await reader.read();
    if (done) break;

    await processChunk(value);
    bytesProcessed += value.length;
    updateProgress(bytesProcessed / file.size);
  }
}
```

10.12 Cross-References

- **Tutorial**: *Learning LARC* Chapter 11 (File Management and OPFS)
- **Components**: Chapter 19 (pan-files, pan-file-picker, pan-image-viewer)
- **Patterns**: Appendix E (File Management Patterns)
- **Related**: Chapter 4 (State Management - IndexedDB), Chapter 12 (Performance)

10.13 Common Issues

10.13.1 Issue: “NotFoundError” when reading files

Problem: File or directory doesn’t exist **Solution**: Check existence with `fileExists()` before reading; handle errors gracefully

10.13.2 Issue: “QuotaExceededError” on write

Problem: Storage quota exceeded **Solution**: Check available space before write; request persistent storage; implement cleanup

10.13.3 Issue: Files lost after browser update

Problem: Non-persistent storage evicted **Solution**: Request persistence via `navigator.storage.persist()`; implement cloud backup

10.13.4 Issue: Slow directory listing with many files

Problem: Synchronous iteration blocks UI **Solution**: Paginate results; use Web Worker for processing; implement virtual scrolling

10.13.5 Issue: OPFS not available in browser

Problem: Older browser or insecure context (non-HTTPS) **Solution**: Feature detect `navigator.storage?.getDirectory`; provide fallback (IndexedDB)

See *Learning LARC* Chapter 11 for complete file management patterns, image processing, and cloud sync strategies.

11 Theming and Styling

Quick reference for theming and styling patterns in LARC applications. For detailed tutorials, see *Learning LARC* Chapter 15.

11.1 Overview

LARC applications use CSS custom properties (variables) for themeable styling, supporting light/dark modes, system preferences, and dynamic theme switching via the PAN bus. The pan-theme-provider component manages global theme state.

Key Concepts:

- CSS custom properties: Runtime-modifiable variables (–color-primary)
- Semantic tokens: Meaningful names (–color-text-primary, not –gray-900)
- Theme provider: Component managing theme state via PAN bus
- System preference: Respect prefers-color-scheme media query
- Scoped themes: Component-specific styling independent of global theme

11.2 Quick Example

```
/* Define theme variables */
:root {
  --color-primary: #3b82f6;
  --color-text: #111827;
  --color-bg: #ffffff;
}

[data-theme="dark"] {
  --color-text: #f9fafb;
  --color-bg: #111827;
}

/* Use in components */
button {
  background: var(--color-primary);
  color: var(--color-bg);
}
```

```
<pan-theme-provider theme="auto"></pan-theme-provider>
<pan-theme-toggle></pan-theme-toggle>
```

11.3 Theme System Structure

11.3.1 Two-Tier Token System

Tier	Purpose	Example
Primitives	Raw color values	`--blue-500: #3b82f6`
Semantic	Meaningful tokens	`--color-primary: var(--blue-500)`

Components use **semantic tokens only**, never primitives. This allows theme changes without updating components.

11.3.2 Standard Theme Variables

```
:root {
  /* Colors */
  --color-primary: #3b82f6;
  --color-text-primary: #111827;
  --color-text-secondary: #6b7280;
  --color-bg-primary: #ffffff;
  --color-bg-secondary: #f9fafb;
  --color-border: #e5e7eb;

  /* Typography */
  --font-family-base: system-ui, sans-serif;
  --font-size-base: 1rem;
  --font-weight-normal: 400;
  --font-weight-bold: 700;
  --line-height-normal: 1.5;

  /* Spacing */
  --space-xs: 0.25rem;
  --space-sm: 0.5rem;
  --space-md: 1rem;
  --space-lg: 1.5rem;
  --space-xl: 2rem;

  /* Borders & Shadows */
  --border-radius: 0.5rem;
  --border-width: 1px;
  --shadow-sm: 0 1px 2px 0 rgba(0,0,0,0.05);
  --shadow-lg: 0 10px 15px -3px rgba(0,0,0,0.1);

  /* Transitions */
  --transition-fast: 150ms ease-in-out;
  --transition-base: 250ms ease-in-out;
```

```
}
```

11.4 Dark Mode Implementation

11.4.1 Manual Dark Mode Override

```
[data-theme="dark"] {
  --color-text-primary: #f9fafb;
  --color-text-secondary: #d1d5db;
  --color-bg-primary: #111827;
  --color-bg-secondary: #1f2937;
  --color-border: #374151;
  --shadow-sm: 0 1px 2px 0 rgba(0,0,0,0.3);
  --shadow-lg: 0 10px 15px -3px rgba(0,0,0,0.5);
}
```

11.4.2 System Preference Detection

```
@media (prefers-color-scheme: dark) {
  :root:not([data-theme="light"]) {
    --color-text-primary: #f9fafb;
    --color-bg-primary: #111827;
    /* ... other dark mode overrides */
  }
}
```

11.5 Theme Provider Component

See Chapter 17 for full API documentation of `pan-theme-provider`.

11.5.1 Basic Usage

```
<pan-theme-provider theme="auto"></pan-theme-provider>
```

11.5.2 JavaScript API

```
const provider = document.querySelector('pan-theme-provider');

// Set theme
provider.setTheme('dark'); // 'light', 'dark', or 'auto'

// Get current theme
const theme = provider.getTheme(); // Returns setting ('auto')
const effective = provider.getEffectiveTheme(); // Returns actual
        ('dark')

// Listen for changes
provider.addEventListener('theme-change', (e) => {
  console.log('Theme:', e.detail.theme);
});
```

11.5.3 PAN Bus Integration

```
import { bus } from '/core/pan-bus.mjs';

// Subscribe to theme changes
bus.subscribe('theme.changed', (msg) => {
  console.log('New theme:', msg.data.effective);
});

// Request theme change
bus.publish('theme.change', { theme: 'dark' });
```

11.6 Theme Toggle Component

```
<!-- Icon button (default) -->
<pan-theme-toggle></pan-theme-toggle>

<!-- Button with label -->
<pan-theme-toggle variant="button" label="Theme"></pan-theme-toggle>

<!-- Dropdown with all options -->
<pan-theme-toggle variant="dropdown"></pan-theme-toggle>
```

11.7 Component-Specific Theming

11.7.1 Shadow DOM Scoping

```
class BrandedCard extends HTMLElement {
  connectedCallback() {
    this.attachShadow({ mode: 'open' });
    this.shadowRoot.innerHTML = `
      <style>
        :host {
          --card-bg: linear-gradient(135deg, #667eea 0%, #764ba2
        100%);
          --card-text: #ffffff;
        }

        .card {
          background: var(--card-bg);
          color: var(--card-text);
          padding: var(--space-lg);
          border-radius: var(--border-radius);
        }

        /* Allow customization */
        :host([variant="flat"]) {
          --card-bg: var(--color-bg-secondary);
          --card-text: var(--color-text-primary);
        }
      </style>
      <div class="card"><slot></slot></div>
    `;
  }
}
```

11.8 Responsive Theming

11.8.1 Viewport-Based Variables

```
:root {
  --space-page: var(--space-md);
  --font-display: var(--font-size-2xl);
}

@media (min-width: 768px) {
  :root {
    --space-page: var(--space-xl);
    --font-display: var(--font-size-3xl);
  }
}

@media (min-width: 1024px) {
  :root {
    --space-page: var(--space-2xl);
    --font-display: 2.5rem;
  }
}

.container {
  padding: var(--space-page);
}
```

11.9 Smooth Theme Transitions

```css
* {
  transition:
    background-color var(--transition-base),
    border-color var(--transition-base),
    color var(--transition-base);
}

/* Disable on page load */
.no-transitions * {
  transition: none !important;
}

/* Respect user preference */
@media (prefers-reduced-motion: reduce) {
  * {
    transition: none !important;
  }
}
```

```javascript
// Disable transitions during initial theme apply
document.documentElement.classList.add('no-transitions');
applyTheme(theme);
requestAnimationFrame(() => {
  document.documentElement.classList.remove('no-transitions');
});
```

11.10 Multi-Brand Support

```
:root {
  --brand-primary: #667eea;
  --brand-logo: url('/logos/default.svg');
}

[data-brand="acme"] {
  --brand-primary: #10b981;
  --brand-logo: url('/logos/acme.svg');
}

[data-brand="techstart"] {
  --brand-primary: #f59e0b;
  --brand-logo: url('/logos/techstart.svg');
}

.brand-button {
  background: var(--brand-primary);
}
```

```
// Switch brands
document.documentElement.setAttribute('data-brand', 'acme');
```

11.11 Accessibility

11.11.1 Contrast Requirements

Maintain WCAG contrast ratios:

- **AA Normal text**: 4.5:1 minimum
- **AA Large text**: 3:1 minimum
- **AAA Normal text**: 7:1 minimum

```
/* Good contrast */
:root {
  --color-text-primary: #111827; /* 16.3:1 on white */
  --color-bg-primary: #ffffff;
}

[data-theme="dark"] {
  --color-text-primary: #f9fafb; /* 17.5:1 on dark */
  --color-bg-primary: #111827;
}
```

11.11.2 Reduced Motion

```
@media (prefers-reduced-motion: reduce) {
  *, *::before, *::after {
    animation-duration: 0.01ms !important;
    animation-iteration-count: 1 !important;
    transition-duration: 0.01ms !important;
  }
}
```

11.11.3 High Contrast Mode

```
@media (prefers-contrast: high) {
  :root {
    --color-text-primary: #000000;
    --color-bg-primary: #ffffff;
    --color-border: #000000;
    --border-width: 2px;
  }
}
```

11.11.4 Screen Reader Announcements

```
function announceThemeChange(theme) {
  const announcement = document.createElement('div');
  announcement.setAttribute('role', 'status');
  announcement.setAttribute('aria-live', 'polite');
  announcement.className = 'sr-only';
  announcement.textContent = `Theme changed to ${theme} mode`;
  document.body.appendChild(announcement);

  setTimeout(() => announcement.remove(), 1000);
}
```

11.12 Performance Tips

Optimization	Benefit
Use data attributes for theme switching	Single change triggers all updates
Avoid deep custom property nesting	Reduces lookup cost
Transition specific properties only	Better performance than `all`
Use CSS containment	Helps browser optimize rendering

11.12.1 Efficient Theme Switching

```
// Good - single attribute change
document.documentElement.setAttribute('data-theme', 'dark');

// Bad - multiple property changes
document.documentElement.style.setProperty('--color-text', '#fff');
document.documentElement.style.setProperty('--color-bg', '#000');
// ... dozens more
```

11.13 Component Reference

See Chapter 17 for complete API documentation:

- **pan-theme-provider**: Global theme management
- **pan-theme-toggle**: Theme switcher UI control

11.14 Cross-References

- **Tutorial**: *Learning LARC* Chapter 15 (Theming and Styling)
- **Components**: Chapter 17 (pan-theme-provider, pan-theme-toggle)
- **Patterns**: Appendix E (Theming Patterns)
- **Related**: Chapter 19 (UI Components)

11.15 Common Issues

11.15.1 Issue: Theme not applying on load

Problem: Flash of unstyled content **Solution**: Apply theme in inline `<script>` before loading components; add `no-transitions` class during initial load

11.15.2 Issue: Custom properties not inheriting

Problem: Shadow DOM doesn't inherit all properties **Solution**: Explicitly pass needed properties through; use CSS `:host` context

11.15.3 Issue: Theme transition lag

Problem: Too many properties transitioning **Solution**: Transition only color-related properties; use `prefers-reduced-motion` to disable

11.15.4 Issue: Dark mode colors look washed out

Problem: Using same colors as light mode **Solution**: Use slightly desaturated colors in dark mode; adjust shadows to be darker

11.15.5 Issue: System preference not detected

Problem: `prefers-color-scheme` not working **Solution**: Ensure HTTPS context; check browser support; verify media query syntax

See *Learning LARC* Chapter 15 for complete theming strategies, advanced CSS patterns, and design system integration.

12 Performance Optimization

Quick reference for performance optimization in LARC applications. For detailed tutorials, see *Learning LARC* Chapter 15.

12.1 Overview

Optimize LARC applications through efficient message patterns, lazy loading, virtual scrolling, and memory management. Performance optimization focuses on user-perceived speed: initial load time, interaction responsiveness, and smooth animations.

Key Concepts:

- Message filtering and throttling to reduce bus overhead
- Lazy loading components and routes for faster initial load
- Virtual scrolling to handle large data sets efficiently
- Debouncing/throttling high-frequency events
- Memory leak prevention through proper cleanup
- Bundle size optimization with tree shaking and code splitting

12.2 Quick Example

```
import { debounce, throttle } from '../utils/timing.js';

class OptimizedSearch extends HTMLElement {
  connectedCallback() {
    // Debounce search input (wait for typing to stop)
    const debouncedSearch = debounce((value) => {
      bus.publish('search.query', { query: value });
    }, 300);

    this.querySelector('input').addEventListener('input', (e) => {
      debouncedSearch(e.target.value);
    });

    // Throttle scroll tracking (limit frequency)
    const throttledScroll = throttle(() => {
      bus.publish('scroll.position', { y: window.scrollY });
    }, 100);

    window.addEventListener('scroll', throttledScroll);
  }
}
```

12.3 Message Bus Optimization

Pattern	Description	Use Case
Specific subscriptions	Subscribe to `user.*` not `*`	Reduce handler invocations
Early returns	Check message relevance first	Skip expensive processing
Unsubscribe aggressively	Unsubscribe when done	Reduce memory and CPU overhead
Throttle publishers	Limit publish frequency (50-100ms)	Mouse/scroll events
Debounce publishers	Wait for pause (300ms)	User input, search
Batch messages	Publish array not individual items	Bulk updates

Throttle vs Debounce:

- **Throttle**: Guarantees function runs at most once per interval (good for scroll)
- **Debounce**: Waits for pause before running (good for search input)

12.4 Timing Utilities

12.4.1 Debounce

```
// utils/debounce.js
export function debounce(fn, delay) {
  let timeoutId = null;

  const debounced = function(...args) {
    clearTimeout(timeoutId);
    timeoutId = setTimeout(() => fn.apply(this, args), delay);
  };

  debounced.cancel = () => clearTimeout(timeoutId);
  return debounced;
}
```

12.4.2 Throttle

```
// utils/throttle.js
export function throttle(fn, interval) {
  let lastCall = 0;

  return function(...args) {
    const now = Date.now();
    if (now - lastCall >= interval) {
      lastCall = now;
      fn.apply(this, args);
    }
  };
}
```

12.4.3 RAF Throttle

```
// utils/raf-throttle.js
export class RAFThrottle {
  constructor(callback) {
    this.callback = callback;
    this.rafId = null;
  }

  trigger(...args) {
    if (this.rafId === null) {
      this.rafId = requestAnimationFrame(() => {
        this.callback(...args);
        this.rafId = null;
      });
    }
  }

  cancel() {
    if (this.rafId) cancelAnimationFrame(this.rafId);
  }
}
```

12.5 Lazy Loading Patterns

12.5.1 Component Lazy Loading

```
// Load component when near viewport
class LazyLoader extends HTMLElement {
  connectedCallback() {
    const componentName = this.getAttribute('component');
    const margin = this.getAttribute('load-distance') || '600px';

    this.observer = new IntersectionObserver(
      async (entries) => {
        if (entries[0].isIntersecting) {
          await import(`./components/${componentName}.mjs`);
          const component = document.createElement(componentName);
          this.replaceWith(component);
        }
      },
      { rootMargin: margin }
    );

    this.observer.observe(this);
  }
}
```

Usage:

```
<lazy-loader component="heavy-chart" load-distance="400px"></lazy-
        loader>
```

12.5.2 Route-Based Code Splitting

```
// app-router.mjs
const routeMap = {
  '/': () => import('./pages/home-page.mjs'),
  '/profile': () => import('./pages/profile-page.mjs'),
  '/settings': () => import('./pages/settings-page.mjs')
};

class AppRouter extends HTMLElement {
  async loadRoute(route) {
    const loader = routeMap[route];
    if (loader) {
      await loader();
      this.innerHTML = `<${this.getComponentName(route)}></>`;
    }
  }
}
```

12.6 Virtual Scrolling

For lists with 1,000+ items, render only visible rows:

```
class VirtualList extends HTMLElement {
  constructor() {
    super();
    this.items = [];
    this.itemHeight = 50;
    this.containerHeight = 600;
    this.scrollTop = 0;
  }

  render() {
    const visibleCount = Math.ceil(this.containerHeight /
        this.itemHeight) + 2;
    const startIndex = Math.floor(this.scrollTop / this.itemHeight);
    const endIndex = Math.min(this.items.length, startIndex +
        visibleCount);

    const visibleItems = this.items.slice(startIndex, endIndex);
    const totalHeight = this.items.length * this.itemHeight;
    const offsetY = startIndex * this.itemHeight;

    this.innerHTML = `
      <div style="height: ${this.containerHeight}px; overflow-y:
        auto;">
        <div style="height: ${totalHeight}px; position: relative;">
          <div style="position: absolute; top: ${offsetY}px;">
            ${visibleItems.map(item =>
        this.renderItem(item)).join('')}
          </div>
        </div>
      </div>
    `;

    this.querySelector('div').addEventListener('scroll', (e) => {
      this.scrollTop = e.target.scrollTop;
```

```
      this.render();
    });
  }

  renderItem(item) {
    return `<div style="height:
        ${this.itemHeight}px;">${item.name}</div>`;
  }
}
```

Usage:

```
<virtual-list item-height="60" container-height="600"></virtual-
        list>
```

See **pan-virtual-list** (Chapter 19) for production-ready virtual list component.

12.7 Memory Management

Pattern	Description	Example
Unsubscribe in `disconnectedCallback`	Clean up PAN subscriptions	`this.unsubscribe()`
Remove event listeners	Clean up DOM listeners	`removeEventListener()`
Clear timers/intervals	Stop periodic tasks	`clearInterval()`
Cancel pending fetches	Abort ongoing requests	`AbortController.abort()`
Use WeakMap for caches	Auto-cleanup with GC	`new WeakMap()`

12.7.1 Cleanup Example

```
class LeakFreeComponent extends HTMLElement {
  connectedCallback() {
    // Subscribe to bus
    this.unsubscribe = bus.subscribe('data.updated',
        this.handleData);

    // Add event listener
    this.handleClick = this.handleClick.bind(this);
    this.addEventListener('click', this.handleClick);

    // Set interval
    this.intervalId = setInterval(() => this.update(), 5000);

    // Fetch with abort controller
    this.abortController = new AbortController();
    fetch('/api/data', { signal: this.abortController.signal });
  }

  disconnectedCallback() {
    // Clean up everything
    if (this.unsubscribe) this.unsubscribe();
    this.removeEventListener('click', this.handleClick);
    clearInterval(this.intervalId);
    this.abortController.abort();
  }
}
```

12.8 Bundle Size Optimization

Technique	Description	Savings
Tree shaking	Remove unused exports	20-40%
Dynamic imports	Load code on demand	Initial: 50-70%
Minification	Compress code	30-50%
Gzip/Brotli	Server compression	70-80%
Dependency auditing	Replace heavy libs	Varies

12.8.1 Bundle Optimization Example

```
// Before: import everything
import moment from 'moment'; // 231 kB

// After: use native APIs
const formatter = new Intl.DateTimeFormat('en-US', {
  year: 'numeric',
  month: 'long',
  day: 'numeric'
});
formatter.format(new Date()); // 0 kB
```

12.8.2 Build Configuration

```
{
  "scripts": {
    "build": "esbuild src/app.js --bundle --minify --splitting --
        outdir=dist"
  }
}
```

12.9 Performance Monitoring

```
class PerformanceMonitor extends HTMLElement {
  connectedCallback() {
    // Navigation timing
    window.addEventListener('load', () => {
      const nav = performance.getEntriesByType('navigation')[0];
      console.log({
        'DNS': nav.domainLookupEnd - nav.domainLookupStart,
        'TCP': nav.connectEnd - nav.connectStart,
        'Request': nav.responseStart - nav.requestStart,
        'Response': nav.responseEnd - nav.responseStart,
        'DOM': nav.domComplete - nav.domLoading,
        'Total': nav.loadEventEnd - nav.fetchStart
      });
    });

    // Long task detection
    new PerformanceObserver((list) => {
      for (const entry of list.getEntries()) {
        if (entry.duration > 50) {
          console.warn(`Long task: ${entry.duration}ms`);
        }
      }
    }).observe({ entryTypes: ['longtask'] });

    // Core Web Vitals
    new PerformanceObserver((list) => {
      for (const entry of list.getEntries()) {
        console.log(`${entry.name}: ${entry.value}`);
      }
    }).observe({ entryTypes: ['largest-contentful-paint', 'first-
        input'] });
  }
```

```
}
```

12.10 Component Reference

See Chapter 19 for performance-optimized UI components:

- **pan-virtual-list**: Production virtual scrolling
- **pan-lazy-image**: Lazy image loading with IntersectionObserver
- **pan-suspense**: Loading states and code splitting

12.11 Complete Example

```
// Optimized data table with virtual scrolling and search
class OptimizedDataTable extends HTMLElement {
  constructor() {
    super();
    this.items = [];
    this.filteredItems = [];
  }

  connectedCallback() {
    // Subscribe to data
    this.unsubscribe = bus.subscribe('table.data', (msg) => {
      this.items = msg.data.items;
      this.filteredItems = this.items;
      this.render();
    });

    // Debounced search
    const debouncedFilter = debounce((query) => {
      this.filteredItems = this.items.filter(item =>
        item.name.toLowerCase().includes(query.toLowerCase())
      );
      this.render();
    }, 300);

    this.innerHTML = `
      <input type="text" placeholder="Search..." />
      <virtual-list item-height="50" container-
        height="600"></virtual-list>
    `;

    this.querySelector('input').addEventListener('input', (e) => {
      debouncedFilter(e.target.value);
```

```
    });
  }

  render() {
    const list = this.querySelector('virtual-list');
    bus.publish('virtual-list.items', { items: this.filteredItems });
  }

  disconnectedCallback() {
    if (this.unsubscribe) this.unsubscribe();
  }
}

customElements.define('optimized-data-table', OptimizedDataTable);
```

12.12 Cross-References

- **Tutorial**: *Learning LARC* Chapter 15 (Performance)
- **Components**: Chapter 19 (pan-virtual-list, pan-lazy-image)
- **Patterns**: Appendix E (Message patterns, optimization strategies)
- **Related**: Chapter 13 (Testing), Chapter 14 (Debugging)

12.13 Common Issues

12.13.1 High CPU usage from message bus

Problem: Too many subscriptions or wildcard patterns
Solution: Use specific topics, unsubscribe aggressively, profile with `pan-debug`

12.13.2 Memory leaks in SPAs

Problem: Components not cleaning up subscriptions/listeners
Solution: Always implement `disconnectedCallback()` cleanup pattern

12.13.3 Slow initial load

Problem: Loading all code upfront
Solution: Lazy load routes/components, code split with dynamic imports

12.13.4 Janky scrolling

Problem: Heavy computations on scroll events
Solution: Use RAF throttle, virtual scrolling for lists, passive event listeners

12.13.5 Large bundle size

Problem: Including entire libraries for small features
Solution: Tree shake, use native APIs, check bundlephobia.com before adding dependencies

13 Testing Strategies

Quick reference for testing LARC applications. For detailed tutorials, see *Learning LARC* Chapter 14.

13.1 Overview

Test LARC applications using unit tests for components, integration tests for message flows, and E2E tests for complete workflows. Message-driven architecture simplifies testing through decoupling and observable side effects.

Key Concepts:

- Testing pyramid: Many unit tests, some integration tests, few E2E tests
- Mock PAN bus for isolated component testing
- Test message flows between components
- Use fake timers for interval/timeout testing
- E2E tests verify complete user workflows in real browsers

13.2 Quick Example

```
import { describe, it, expect, beforeEach } from 'vitest';
import { CounterButton } from '../components/counter-button.mjs';

describe('CounterButton', () => {
  let element;

  beforeEach(() => {
    element = document.createElement('counter-button');
    document.body.appendChild(element);
  });

  it('should increment count when clicked', () => {
    const button = element.querySelector('button');

    expect(element.count).toBe(0);
    button.click();
    expect(element.count).toBe(1);
    button.click();
    expect(element.count).toBe(2);
  });
});
```

13.3 Test Environment Setup

13.3.1 Vitest Configuration

```
// vitest.config.js
import { defineConfig } from 'vitest/config';

export default defineConfig({
  test: {
    environment: 'happy-dom',
    globals: true,
    setupFiles: ['./tests/setup.js']
  }
});
```

13.3.2 Test Setup

```
// tests/setup.js
import { beforeEach, afterEach } from 'vitest';
import { MockBus } from './mocks/mock-bus.js';

let mockBus;

beforeEach(() => {
  mockBus = new MockBus();
  global.publish = mockBus.publish.bind(mockBus);
  global.subscribe = mockBus.subscribe.bind(mockBus);
});

afterEach(() => {
  mockBus.reset();
  document.body.innerHTML = '';
});

export function getMockBus() {
  return mockBus;
}
```

13.4 Mock PAN Bus

```
// tests/mocks/mock-bus.js
class MockBus {
  constructor() {
    this.subscriptions = new Map();
    this.published = [];
  }

  publish(topic, data) {
    this.published.push({ topic, data, timestamp: Date.now() });

    // Trigger subscriptions
    const handlers = this.subscriptions.get(topic) || [];
    handlers.forEach(handler => handler({ topic, data }));

    // Trigger wildcard subscriptions
    this.getWildcardHandlers(topic).forEach(handler => {
      handler({ topic, data });
    });
  }

  subscribe(pattern, handler) {
    if (!this.subscriptions.has(pattern)) {
      this.subscriptions.set(pattern, []);
    }

    this.subscriptions.get(pattern).push(handler);

    // Return unsubscribe function
    return () => {
      const handlers = this.subscriptions.get(pattern);
      const index = handlers.indexOf(handler);
      if (index > -1) handlers.splice(index, 1);
```

```
    };
  }

  getWildcardHandlers(topic) {
    const handlers = [];
    for (const [pattern, patternHandlers] of this.subscriptions) {
      if (this.matchesPattern(topic, pattern)) {
        handlers.push(...patternHandlers);
      }
    }
    return handlers;
  }

  matchesPattern(topic, pattern) {
    if (pattern === '*') return true;
    if (pattern === topic) return false;

    const patternParts = pattern.split('.');
    const topicParts = topic.split('.');

    if (patternParts.length !== topicParts.length) return false;

    return patternParts.every((part, i) =>
      part === '*' || part === topicParts[i]
    );
  }

  reset() {
    this.subscriptions.clear();
    this.published = [];
  }

  // Test helpers
```

```
  getPublished(topic) {
    return this.published.filter(msg => msg.topic === topic);
  }

  getLastPublished(topic) {
    const messages = this.getPublished(topic);
    return messages[messages.length - 1];
  }

  wasPublished(topic, data) {
    return this.published.some(msg =>
      msg.topic === topic &&
      JSON.stringify(msg.data) === JSON.stringify(data)
    );
  }
}

export { MockBus };
```

13.5 Unit Testing Patterns

Pattern	Description	Example
Component creation	Create and mount component	`document.createElement('my-component')`
Attribute testing	Test attribute changes	`element.setAttribute('value', 'test')`
Event testing	Simulate user interactions	`button.click()`
Spy methods	Verify method calls	`vi.spyOn(element, 'method')`
Mock fetch	Stub network requests	`global.fetch = vi.fn()`

13.5.1 Testing Attributes

```
it('should update when attributes change', () => {
  element.setAttribute('username', 'Alice');
  expect(element.textContent).toContain('Alice');

  element.setAttribute('username', 'Bob');
  expect(element.textContent).toContain('Bob');
  expect(element.textContent).not.toContain('Alice');
});
```

13.5.2 Testing Message Publishing

```
it('should publish notification', () => {
  const mockBus = getMockBus();

  element.showNotification('Hello');

  const published = mockBus.getLastPublished('notification.show');
  expect(published).toBeDefined();
  expect(published.data.message).toBe('Hello');
});
```

13.5.3 Testing Message Subscriptions

```
it('should update on message', () => {
  publish('data.updated', { value: 42 });

  expect(element.value).toBe(42);
  expect(element.textContent).toContain('42');
});
```

13.6 Integration Testing

Test multiple components communicating via PAN bus:

```
describe('Shopping Cart Integration', () => {
  let catalog, cart, badge;

  beforeEach(() => {
    catalog = document.createElement('product-catalog');
    cart = document.createElement('shopping-cart');
    badge = document.createElement('cart-badge');

    document.body.append(catalog, cart, badge);
  });

  it('should update cart and badge when product added', () => {
    publish('cart.item.added', {
      productId: 1,
      name: 'Widget',
      price: 10,
      quantity: 1
    });

    expect(cart.items).toHaveLength(1);
    expect(cart.items[0].name).toBe('Widget');
    expect(badge.itemCount).toBe(1);
  });

  it('should publish cart.updated message', () => {
    const mockBus = getMockBus();

    publish('cart.item.added', {
      productId: 1,
      name: 'Widget',
      price: 10,
      quantity: 1
```

```
    });

    const updated = mockBus.getLastPublished('cart.updated');
    expect(updated.data.items).toHaveLength(1);
    expect(updated.data.total).toBe(10);
  });
});
```

13.7 Testing Async Operations

13.7.1 Testing Promises

```
it('should load data', async () => {
  global.fetch = vi.fn().mockResolvedValueOnce({
    json: async () => ({ items: [1, 2, 3] })
  });

  const element = document.createElement('data-loader');
  document.body.appendChild(element);

  await vi.waitFor(() => {
    const loaded = getMockBus().getLastPublished('data.loaded');
    expect(loaded).toBeDefined();
    expect(loaded.data.items).toEqual([1, 2, 3]);
  });
});
```

13.7.2 Testing Errors

```
it('should handle fetch errors', async () => {
  global.fetch = vi.fn().mockRejectedValueOnce(
    new Error('Network error')
  );

  const element = document.createElement('data-loader');
  document.body.appendChild(element);

  await new Promise(resolve => setTimeout(resolve, 0));

  const error = getMockBus().getLastPublished('data.error');
  expect(error.data.error).toBe('Network error');
});
```

13.7.3 Testing Timers

```
it('should save at intervals', () => {
  vi.useFakeTimers();

  const element = document.createElement('auto-saver');
  document.body.appendChild(element);

  vi.advanceTimersByTime(5000);
  expect(getMockBus().getPublished('data.save')).toHaveLength(1);

  vi.advanceTimersByTime(5000);
  expect(getMockBus().getPublished('data.save')).toHaveLength(2);

  vi.useRealTimers();
});
```

13.8 E2E Testing with Playwright

13.8.1 Setup

```
npm install -D @playwright/test
npx playwright install
```

13.8.2 E2E Test Example

```
// tests/e2e/shopping-cart.spec.js
import { test, expect } from '@playwright/test';

test.describe('Shopping Cart', () => {
  test.beforeEach(async ({ page }) => {
    await page.goto('http://localhost:3000');
  });

  test('should add item to cart', async ({ page }) => {
    await page.click('button:has-text("Add to Cart")');

    const badge = page.locator('cart-badge');
    await expect(badge).toContainText('1');

    const cart = page.locator('shopping-cart');
    await expect(cart).toContainText('Widget');
    await expect(cart).toContainText('$10');
  });

  test('should persist cart after reload', async ({ page }) => {
    await page.click('button:has-text("Add to Cart")');
    await page.reload();

    const cart = page.locator('shopping-cart');
    await expect(cart).toContainText('Widget');
  });
});
```

13.8.3 Testing Theme Switching

```
test('should toggle dark mode', async ({ page }) => {
  await page.goto('http://localhost:3000');

  const html = page.locator('html');
  await expect(html).toHaveAttribute('data-theme', 'light');

  await page.click('button:has-text("Dark")');
  await expect(html).toHaveAttribute('data-theme', 'dark');

  await page.reload();
  await expect(html).toHaveAttribute('data-theme', 'dark');
});
```

13.9 Test Utilities

13.9.1 Component Test Harness

```
// tests/utils/component-harness.js
class ComponentHarness {
  constructor(tagName, attributes = {}) {
    this.element = document.createElement(tagName);

    for (const [key, value] of Object.entries(attributes)) {
      this.element.setAttribute(key, value);
    }

    document.body.appendChild(this.element);
  }

  query(selector) {
    return this.element.querySelector(selector);
  }

  text() {
    return this.element.textContent.trim();
  }

  click(selector) {
    const el = selector ? this.query(selector) : this.element;
    el.click();
    return this;
  }

  type(selector, value) {
    const input = this.query(selector);
    input.value = value;
    input.dispatchEvent(new Event('input', { bubbles: true }));
    return this;
  }
```

```
  async waitFor(condition, timeout = 1000) {
    const start = Date.now();

    while (Date.now() - start < timeout) {
      if (condition(this.element)) return;
      await new Promise(resolve => setTimeout(resolve, 50));
    }

    throw new Error('Timeout waiting for condition');
  }

  destroy() {
    this.element.remove();
  }
}

export { ComponentHarness };
```

Usage:

```
const harness = new ComponentHarness('user-profile', { 'user-id':
        '123' });
await harness.waitFor(el => el.textContent.includes('Alice'));
expect(harness.text()).toContain('Alice');
harness.destroy();
```

13.9.2 Message Helper

```
// tests/utils/message-helper.js
class MessageHelper {
  constructor() {
    this.bus = getMockBus();
  }

  async publishAndWait(publishTopic, publishData, waitTopic, timeout
        = 1000) {
    return new Promise((resolve, reject) => {
      const timeoutId = setTimeout(() => {
        unsubscribe();
        reject(new Error(`Timeout waiting for ${waitTopic}`));
      }, timeout);

      const unsubscribe = subscribe(waitTopic, (msg) => {
        clearTimeout(timeoutId);
        unsubscribe();
        resolve(msg.data);
      });

      publish(publishTopic, publishData);
    });
  }

  assertPublished(topic, data = null) {
    const messages = this.bus.getPublished(topic);

    if (messages.length === 0) {
      throw new Error(`Expected message on topic "${topic}"`);
    }

    if (data !== null) {
```

```
      const match = messages.some(msg =>
        JSON.stringify(msg.data) === JSON.stringify(data)
      );

      if (!match) {
        throw new Error(`Expected message with data
        ${JSON.stringify(data)}`);
      }
    }
  }
}

export { MessageHelper };
```

Usage:

```
const helper = new MessageHelper();

const response = await helper.publishAndWait(
  'data.request',
  { id: 123 },
  'data.response'
);

expect(response.id).toBe(123);
helper.assertPublished('data.request', { id: 123 });
```

13.10 Test Coverage

Run tests with coverage:

```
npm install -D @vitest/coverage-v8
npx vitest --coverage
```

Coverage Targets:

- Overall: 80%+
- Critical paths (auth, payments): 100%
- UI glue code: 50-70% acceptable

13.11 Testing Checklist

Area	Unit Tests	Integration Tests	E2E Tests
Components render correctly	✓		
Attributes update DOM	✓		
Messages published	✓		
Messages received	✓		
Multi-component flows		✓	
Message patterns		✓	
User workflows			✓
Persistence			✓
Theme switching			✓

13.12 Complete Example

```
// Full test suite for notification system
import { describe, it, expect, beforeEach, vi } from 'vitest';
import { NotificationDisplay } from '../components/notification-
        display.mjs';
import { NotificationService } from '../services/notification-
        service.mjs';
import { getMockBus } from './setup.js';

describe('Notification System', () => {
  let display, service, mockBus;

  beforeEach(() => {
    mockBus = getMockBus();

    display = document.createElement('notification-display');
    service = new NotificationService();

    document.body.appendChild(display);
  });

  it('should show notification', () => {
    service.show('Hello', 'info');

    expect(display.notifications).toHaveLength(1);
    expect(display.textContent).toContain('Hello');
  });

  it('should auto-dismiss after timeout', async () => {
    vi.useFakeTimers();

    service.show('Temporary', 'info', { duration: 3000 });

    expect(display.notifications).toHaveLength(1);
```

```
    vi.advanceTimersByTime(3000);

    await vi.waitFor(() => {
      expect(display.notifications).toHaveLength(0);
    });

    vi.useRealTimers();
  });

  it('should handle multiple notifications', () => {
    service.show('First', 'info');
    service.show('Second', 'warning');
    service.show('Third', 'error');

    expect(display.notifications).toHaveLength(3);
    expect(mockBus.getPublished('notification.show')).toHaveLength(3);

  });
});
```

13.13 Component Reference

See Chapter 17 (pan-bus) for testing message patterns.

13.14 Cross-References

- **Tutorial**: *Learning LARC* Chapter 14 (Testing)
- **Components**: Chapter 17 (pan-bus testing helpers)
- **Patterns**: Appendix E (Test patterns)

- **Related**: Chapter 12 (Performance), Chapter 14 (Debugging)

13.15 Common Issues

13.15.1 Tests failing intermittently

Problem: Race conditions in async tests
Solution: Use `vi.waitFor()` or `await` properly, avoid fixed timeouts

13.15.2 Mock bus not working

Problem: Components using global `publish` / `subscribe` before mock setup
Solution: Import components after mock setup in `beforeEach`, use dynamic imports

13.15.3 E2E tests timing out

Problem: Waiting for elements that never appear
Solution: Use Playwright's auto-waiting, check network requests, verify app is running

13.15.4 Coverage not including files

Problem: Files not imported by tests
Solution: Add files to test suite or use coverage include patterns

13.15.5 Memory leaks in tests

Problem: Components not cleaned up
Solution: Always remove components in `afterEach`, clear timers with `vi.useRealTimers()`

14 Error Handling and Debugging

Quick reference for error handling and debugging in LARC applications. For detailed tutorials, see *Learning LARC* Chapter 14.

14.1 Overview

Handle errors gracefully through component-level error boundaries, centralized error monitoring, and message tracing. LARC's isolated component architecture naturally contains errors, preventing cascading failures.

Key Concepts:

- Error boundaries contain failures within components
- Global error handlers monitor application health
- Message tracing tracks pub/sub flows
- Structured logging provides debugging context
- DevTools integration for professional debugging

14.2 Quick Example

```
class ResilientComponent extends HTMLElement {
  connectedCallback() {
    this.unsubscribe = bus.subscribe('data.fetch', async (msg) => {
      try {
        const data = await this.fetchData(msg.data.id);
        bus.publish('data.loaded', { data });
      } catch (error) {
        bus.publish('app.error', {
          component: 'ResilientComponent',
          error: error.message,
          context: { id: msg.data.id }
        });
        this.showError(error);
      }
    });
  }

  showError(error) {
    this.innerHTML = `
      <div class="error">
        Error: ${error.message}
        <button onclick="location.reload()">Retry</button>
      </div>
    `;
  }
}
```

14.3 Error Handling Patterns

Pattern	Description	Use Case
Try-catch blocks	Catch synchronous errors	Immediate operations
Promise `.catch()`	Handle async errors	Fetch, timeouts
Error boundaries	Contain component failures	Prevent cascading failures
Global error handler	Catch unhandled errors	Last line of defense
Error messages	Publish via PAN bus	Centralized monitoring

14.3.1 Component Error Boundary

```
class ErrorBoundary extends HTMLElement {
  constructor() {
    super();
    this.error = null;
  }

  connectedCallback() {
    this.renderContent();

    // Catch errors from child components
    this.addEventListener('error', (e) => {
      this.error = e.error || e;
      this.renderContent();
      e.stopPropagation();
    });
  }

  renderContent() {
    if (this.error) {
      this.innerHTML = `
        <div class="error-boundary">
          <h3>Something went wrong</h3>
          <p>${this.error.message}</p>
          <button onclick="location.reload()">Reload</button>
        </div>
      `;
    }
  }
}
```

14.3.2 Global Error Monitor

```
class ErrorMonitor extends HTMLElement {
  constructor() {
    super();
    this.errors = [];
  }

  connectedCallback() {
    // Subscribe to error messages
    this.unsubscribe = bus.subscribe('app.error', (msg) => {
      this.logError(msg.data);
    });

    // Catch global errors
    window.addEventListener('error', (e) => {
      this.logError({
        message: e.message,
        source: e.filename,
        line: e.lineno
      });
    });

    // Catch promise rejections
    window.addEventListener('unhandledrejection', (e) => {
      this.logError({
        message: e.reason?.message || 'Promise rejected',
        stack: e.reason?.stack
      });
    });
  }

  logError(error) {
    const entry = {
```

```
      ...error,
      timestamp: new Date().toISOString(),
      url: location.href
    };

    this.errors.push(entry);
    console.error('[App Error]', entry);

    // Send to error tracking service
    this.reportError(entry);
  }

  reportError(error) {
    if (window.errorTracker) {
      window.errorTracker.captureException(error);
    }
  }
}
```

14.4 Message Tracing

14.4.1 Enable Bus Debugging

```
<pan-bus debug="true" enable-tracing="true"></pan-bus>
```

14.4.2 Custom Message Tracer

```
class MessageTracer extends HTMLElement {
  constructor() {
    super();
    this.log = [];
    this.tracing = false;
  }

  connectedCallback() {
    this.unsubscribe = bus.subscribe('*', (msg) => {
      if (!this.tracing) return;

      this.log.push({
        topic: msg.topic,
        data: msg.data,
        timestamp: performance.now()
      });

      console.log(
        `%c[MSG] ${msg.topic}`,
        'color: blue; font-weight: bold',
        msg.data
      );
    });
  }

  startTracing() {
    this.tracing = true;
    this.log = [];
  }

  stopTracing() {
    this.tracing = false;
```

```
    return this.log;
  }

  exportLog() {
    const blob = new Blob([JSON.stringify(this.log, null, 2)], {
      type: 'application/json'
    });
    const url = URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = url;
    a.download = `message-trace-${Date.now()}.json`;
    a.click();
  }
}
```

14.5 Structured Logging

```
class Logger {
  constructor() {
    this.level = 'info'; // debug, info, warn, error
  }

  log(level, message, context = {}) {
    if (!this.shouldLog(level)) return;

    const entry = {
      timestamp: new Date().toISOString(),
      level,
      message,
      context,
      sessionId: this.getSessionId()
    };

    this.output(entry);
    this.send(entry);
  }

  shouldLog(level) {
    const levels = ['debug', 'info', 'warn', 'error'];
    return levels.indexOf(level) >= levels.indexOf(this.level);
  }

  output(entry) {
    const styles = {
      debug: 'color: gray',
      info: 'color: blue',
      warn: 'color: orange',
      error: 'color: red; font-weight: bold'
    };
```

```
    console.log(
      `%c[${entry.level.toUpperCase()}] ${entry.message}`,
      styles[entry.level],
      entry.context
    );
  }

  send(entry) {
    if (entry.level === 'error' || entry.level === 'warn') {
      navigator.sendBeacon('/api/logs', JSON.stringify(entry));
    }
  }

  getSessionId() {
    return sessionStorage.getItem('sessionId') || 'unknown';
  }

  debug(message, context) { this.log('debug', message, context); }
  info(message, context) { this.log('info', message, context); }
  warn(message, context) { this.log('warn', message, context); }
  error(message, context) { this.log('error', message, context); }
}

// Global logger instance
const logger = new Logger();
export { logger };
```

Usage:

```
import { logger } from './logger.js';

class UserService {
  async login(username, password) {
    logger.info('User login attempt', { username });

    try {
      const response = await fetch('/api/login', {
        method: 'POST',
        body: JSON.stringify({ username, password })
      });

      if (!response.ok) {
        throw new Error('Login failed');
      }

      const data = await response.json();
      logger.info('User logged in', { username, userId: data.userId
        });
      return data;
    } catch (error) {
      logger.error('Login failed', { username, error: error.message
        });
      throw error;
    }
  }
}
```

14.6 DevTools Integration

14.6.1 Custom Console Formatters

```
// Enable custom formatters in Chrome DevTools
if (window.devtoolsFormatters) {
  window.devtoolsFormatters.push({
    header(obj) {
      if (!obj?.__larcMessage) return null;

      return ['div', { style: 'color: #00f; font-weight: bold' },
        `[LARC] ${obj.topic}`
      ];
    },
    hasBody(obj) {
      return obj?.__larcMessage;
    },
    body(obj) {
      return ['div', {},
        ['div', {}, `Topic: ${obj.topic}`],
        ['div', {}, `Data: `, ['object', { object: obj.data }]],
        ['div', {}, `Time: ${new
        Date(obj.timestamp).toISOString()}`]
      ];
    }
  });
}
```

14.6.2 Performance Monitoring

```
class PerformanceMonitor extends HTMLElement {
  connectedCallback() {
    this.unsubscribe = bus.subscribe('*', (msg) => {
      const mark = `msg-${msg.topic}-${Date.now()}`;
      performance.mark(mark);

      setTimeout(() => {
        performance.measure(msg.topic, mark);
        const entries = performance.getEntriesByType('measure');
        const latest = entries[entries.length - 1];

        if (latest.duration > 16) { // Slower than 60fps
          console.warn(`Slow handler: ${msg.topic} took
        ${latest.duration}ms`);
          bus.publish('performance.warning', {
            topic: msg.topic,
            duration: latest.duration
          });
        }

        performance.clearMarks(mark);
        performance.clearMeasures(msg.topic);
      }, 0);
    });
  }
}
```

14.7 Common Pitfalls

14.7.1 Message Type Typos

Problem: Misspelled message topics
Solution: Use constants

```
// messages.js
export const MSG = {
  USER_LOGIN: 'user.login',
  USER_LOGOUT: 'user.logout',
  CART_UPDATE: 'cart.update'
};

// Usage
import { MSG } from './messages.js';

bus.publish(MSG.USER_LOGIN, { userId: 123 });
bus.subscribe(MSG.USER_LOGIN, (msg) => { /* works */ });
```

14.7.2 Infinite Message Loops

Problem: A → B → A → B forever
Solution: Loop detection

```
class LoopDetector extends HTMLElement {
  constructor() {
    super();
    this.stack = [];
  }

  connectedCallback() {
    this.unsubscribe = bus.subscribe('*', (msg) => {
      this.stack.push(msg.topic);

      const recent = this.stack.slice(-5);
      const counts = {};
      recent.forEach(t => counts[t] = (counts[t] || 0) + 1);

      if (Object.values(counts).some(c => c >= 3)) {
        console.error('Message loop detected:', recent);
        bus.publish('system.loop-detected', { sequence: recent });
      }

      setTimeout(() => this.stack.shift(), 1000);
    });
  }
}
```

14.7.3 Async Race Conditions

Problem: Multiple overlapping async operations

Solution: Request ID tracking

```
class DataLoader extends HTMLElement {
  constructor() {
    super();
    this.requestId = 0;
  }

  async loadData(id) {
    const currentRequest = ++this.requestId;

    try {
      const data = await fetch(`/api/data/${id}`).then(r =>
        r.json());

      // Only update if still the latest request
      if (currentRequest === this.requestId) {
        this.data = data;
        this.render();
      }
    } catch (error) {
      if (currentRequest === this.requestId) {
        this.showError(error);
      }
    }
  }
}
```

14.7.4 Stale Closures

Problem: Handler captures old state
Solution: Always access `this.state` directly

```
// Bad
class BadCounter extends HTMLElement {
  connectedCallback() {
    const count = this.count; // Captured at registration time
    this.unsubscribe = bus.subscribe('log', () => {
      console.log(count); // Always logs initial value
    });
  }
}

// Good
class GoodCounter extends HTMLElement {
  connectedCallback() {
    this.unsubscribe = bus.subscribe('log', () => {
      console.log(this.count); // Always current value
    });
  }
}
```

14.8 Debugging Checklist

When something goes wrong:

1. **Is the message being sent?**
   - Add `console.log()` before `publish()`
   - Check MessageTracer
2. **Is the topic correct?**
   - Use message constants
   - Check for typos
3. **Is the handler registered?**
   - Verify `subscribe()` is called

    - Check component is mounted
4. **Is the handler called?**
    - Add breakpoint or `console.log()`
    - Check wildcard patterns
5. **Is state updating?**
    - Log before/after changes
    - Check property names
6. **Is render triggered?**
    - Verify render method runs
    - Check for errors in render
7. **Are there async issues?**
    - Use request IDs
    - Check Promise handling
8. **Is there a loop?**
    - Use LoopDetector
    - Review message flow

14.9 Component Reference

See **pan-debug** (Chapter 21) for browser-based debugging tools.

14.10 Complete Example

```
// Full error handling setup
class App extends HTMLElement {
  connectedCallback() {
    // Global error monitor
    const errorMonitor = document.createElement('error-monitor');
    document.body.appendChild(errorMonitor);

    // Message tracer (dev only)
    if (import.meta.env.DEV) {
      const tracer = document.createElement('message-tracer');
      tracer.startTracing();
      document.body.appendChild(tracer);
    }

    // Loop detector
    const loopDetector = document.createElement('loop-detector');
    document.body.appendChild(loopDetector);

    // Performance monitor
    const perfMonitor = document.createElement('performance-
        monitor');
    document.body.appendChild(perfMonitor);

    // Subscribe to errors
    this.unsubscribe = bus.subscribe('app.error', (msg) => {
      this.showErrorNotification(msg.data);
    });
  }

  showErrorNotification(error) {
    const notification = document.createElement('div');
    notification.className = 'error-notification';
```

```
    notification.textContent = error.message;
    document.body.appendChild(notification);

    setTimeout(() => notification.remove(), 5000);
  }
}
```

14.11 Cross-References

- **Tutorial**: *Learning LARC* Chapter 14 (Error Handling & Debugging)
- **Components**: Chapter 21 (pan-debug)
- **Patterns**: Appendix E (Error patterns)
- **Related**: Chapter 12 (Performance), Chapter 13 (Testing)

14.12 Common Issues

14.12.1 Errors not caught

Problem: Errors in async code not handled
Solution: Always use try-catch or `.catch()`, add global error handlers

14.12.2 Missing stack traces

Problem: Error stack is lost
Solution: Preserve `error.stack` when re-throwing or logging

14.12.3 Too much logging in production

Problem: Console spam, performance impact
Solution: Set log level to ‘warn’ or ‘error’ in production, use conditional logging

14.12.4 Can’t reproduce bug

Problem: Error only happens for some users
Solution: Add user context to logs, enable remote error tracking (Sentry, LogRocket)

14.12.5 Debug mode left on

Problem: Debug code in production
Solution: Use environment variables, build-time dead code elimination

15 Advanced Patterns

Quick reference for advanced LARC architecture patterns. For detailed tutorials, see *Learning LARC* Chapter 7.

15.1 Overview

Advanced patterns for complex scenarios: message bridging, multi-bus architectures, backend integration, micro-frontends, and plugin systems. These patterns solve scalability and modularity challenges in large applications.

Key Concepts: - Message bridging between buses - Domain-segregated architectures - API gateway patterns - Micro-frontend integration - Plugin/middleware systems

15.2 Quick Example

```
// Bidirectional bridge between two buses
class BidirectionalBridge {
  constructor(busA, busB, config) {
    this.busA = busA;
    this.busB = busB;

    // Forward A → B
    config.aToB.forEach(type => {
      busA.subscribe(type, msg => busB.publish(type, msg.data));
    });

    // Forward B → A
    config.bToA.forEach(type => {
      busB.subscribe(type, msg => busA.publish(type, msg.data));
    });
  }
}

// Usage
const bridge = new BidirectionalBridge(mainBus, widgetBus, {
  aToB: ['theme.changed', 'user.login'],
  bToA: ['widget.action']
});
```

15.3 Message Bridging Patterns

Pattern	Use Case	Key Features
Basic Forward	One-way message relay	Simple topic mapping
Bidirectional	Two-way communication	Transform and filter support
Translation	Protocol conversion	Type and data transformation
Hierarchical	Parent-child buses	Bubble-up and propagate-down

15.4 Backend Integration Patterns

Pattern	Use Case	Key Features
API Gateway	Centralized HTTP calls	Request queue, offline support
WebSocket Bridge	Real-time bidirectional	Auto-reconnect, message queue
GraphQL Client	Structured queries	Response caching, mutation invalidation

15.5 Common Patterns

15.5.1 Pattern 1: API Gateway Component

Centralize all API calls with offline support.

```
class APIGateway extends HTMLElement {
  connectedCallback() {
    const bus = document.querySelector('pan-bus');

    bus.subscribe('api.request', async (msg) => {
      const { method, endpoint, data, requestId } = msg.data;

      try {
        const response = await fetch(`/api${endpoint}`, {
          method,
          headers: {
            'Content-Type': 'application/json',
            'Authorization': `Bearer ${this.token}`
          },
          body: data ? JSON.stringify(data) : undefined
        });

        if (!response.ok) throw new Error(`HTTP
        ${response.status}`);

        const result = await response.json();
        bus.publish('api.success', { requestId, result });

      } catch (error) {
        bus.publish('api.error', { requestId, error: error.message
        });
      }
    });
  }
}

customElements.define('api-gateway', APIGateway);
```

Usage:

```
// In any component
bus.publish('api.request', {
  method: 'GET',
  endpoint: '/users/123',
  requestId: crypto.randomUUID()
});

bus.subscribe('api.success', (msg) => {
  if (msg.data.requestId === myRequestId) {
    this.user = msg.data.result;
  }
});
```

15.5.2 Pattern 2: Micro-Frontend Loader

Load remote modules dynamically.

```
class MicroFrontendLoader extends HTMLElement {
  async loadModule(name, url, props) {
    try {
      const module = await import(/* webpackIgnore: true */ url);
      const instance = new module.default(props);

      this.modules.set(name, instance);
      this.dispatchEvent(new CustomEvent('module-loaded', {
        detail: { name }
      }));

    } catch (error) {
      console.error(`Failed to load module ${name}:`, error);
    }
  }

  unloadModule(name) {
    const module = this.modules.get(name);
    if (module?.destroy) module.destroy();
    this.modules.delete(name);
  }
}

// Usage
const loader = document.querySelector('mf-loader');
await loader.loadModule('cart', 'https://cdn.example.com/cart.js', {
  bus: document.querySelector('pan-bus'),
  apiEndpoint: '/api/cart'
});
```

15.5.3 Pattern 3: Plugin Registry with Hooks

Extensible plugin system.

```
class PluginRegistry {
  constructor(bus) {
    this.bus = bus;
    this.plugins = new Map();
    this.hooks = new Map();
  }

  register(id, plugin) {
    // Initialize plugin
    plugin.init({ bus: this.bus });

    // Register hooks
    if (plugin.hooks) {
      Object.entries(plugin.hooks).forEach(([hookName, handler]) =>
        {
        if (!this.hooks.has(hookName)) {
          this.hooks.set(hookName, []);
        }
        this.hooks.get(hookName).push({ id, handler });
      });
    }

    this.plugins.set(id, plugin);
  }

  async executeHook(hookName, data) {
    const handlers = this.hooks.get(hookName) || [];
    let result = data;

    for (const { handler } of handlers) {
      result = await handler(result);
    }
```

```
    return result;
  }
}

// Plugin example
const analyticsPlugin = {
  init({ bus }) {
    bus.subscribe('*', (msg) => {
      gtag('event', msg.topic, msg.data);
    });
  },

  hooks: {
    'before-submit': (formData) => {
      console.log('Analytics: Form submit', formData);
      return formData;
    }
  }
};

// Usage
const registry = new PluginRegistry(bus);
registry.register('analytics', analyticsPlugin);

const data = await registry.executeHook('before-submit', formData);
```

15.6 Architecture Tables

15.6.1 Multi-Bus Architecture

Approach	Pros	Cons	Use Case
Domain-Segregated	Clear boundaries, testable	More complexity	Large apps with distinct domains
Hierarchical	Natural parent-child	Bubble/ propagate overhead	Nested module systems
Federated	Independent deployment	Coordination needed	Micro-frontends

15.6.2 Middleware Patterns

Type	Use Case	Example
Logging	Debug message flow	Log all messages
Rate Limiting	Prevent spam	Max 10 msgs/sec/topic
Transform	Add metadata	Inject timestamp, userId
Auth	Protect actions	Check token before API calls
Error Handling	Catch failures	Try/catch middleware chain

15.7 Complete Example

Multi-domain e-commerce app with API gateway, WebSocket, and plugin system.

```
// 1. Create domain buses
const buses = {
  auth: document.createElement('pan-bus'),
  cart: document.createElement('pan-bus'),
  ui: document.createElement('pan-bus')
};

Object.values(buses).forEach(bus => document.body.appendChild(bus));

// 2. Bridge auth events to UI
buses.auth.subscribe('user.login', (msg) => {
  buses.ui.publish('notification', {
    message: `Welcome, ${msg.data.username}!`
  });
});

// 3. API Gateway on auth bus
class AuthAPI extends HTMLElement {
  connectedCallback() {
    const bus = document.querySelector('pan-bus[domain="auth"]');

    bus.subscribe('auth.login', async (msg) => {
      const response = await fetch('/api/login', {
        method: 'POST',
        body: JSON.stringify(msg.data)
      });

      if (response.ok) {
        const { token, user } = await response.json();
        localStorage.setItem('token', token);
        bus.publish('user.login', user);
      }
```

```
    });
  }
}

customElements.define('auth-api', AuthAPI);

// 4. WebSocket bridge for cart updates
class CartSocket extends HTMLElement {
  connectedCallback() {
    this.ws = new WebSocket('wss://example.com/cart');
    const bus = document.querySelector('pan-bus[domain="cart"]');

    this.ws.onmessage = (event) => {
      const update = JSON.parse(event.data);
      bus.publish('cart.updated', update);
    };

    bus.subscribe('cart.add-item', (msg) => {
      this.ws.send(JSON.stringify({
        action: 'add',
        item: msg.data
      }));
    });
  }
}

customElements.define('cart-socket', CartSocket);

// 5. Plugin system
const registry = new PluginRegistry(buses.ui);

registry.register('analytics', {
  init({ bus }) {
```

```
    bus.subscribe('*', (msg) => {
      gtag('event', msg.topic, msg.data);
    });
  }
});

registry.register('error-handler', {
  init({ bus }) {
    bus.subscribe('app.error', (msg) => {
      Sentry.captureException(msg.data.error);
    });
  }
});
```

15.8 Component Reference

See Chapter 17 for core PAN Bus API documentation.

15.9 Cross-References

- **Tutorial**: *Learning LARC* Chapter 7 (Advanced Patterns)
- **Components**: Chapter 17 (pan-bus), Chapter 20 (integration components)
- **Backend Integration**: Chapter 7 (Data Fetching), Chapter 9 (Realtime Features)
- **Testing**: Chapter 13 (mocking bridges and plugins)

15.10 Common Issues

15.10.1 Bridge Message Loops

Problem: Messages bounce between bridges infinitely
Solution: Add loop detection with message IDs or use one-way bridges

```
const processedIds = new Set();

bus.subscribe('*', (msg) => {
  if (processedIds.has(msg.id)) return;
  processedIds.add(msg.id);

  // Forward to other bus...

  // Clean up old IDs
  if (processedIds.size > 1000) {
    processedIds.clear();
  }
});
```

15.10.2 Plugin Hook Failures

Problem: One plugin failure breaks entire hook chain
Solution: Wrap hooks in try/catch

```
async executeHook(hookName, data) {
  const handlers = this.hooks.get(hookName) || [];
  let result = data;

  for (const { id, handler } of handlers) {
    try {
      result = await handler(result);
    } catch (error) {
      console.error(`Plugin ${id} hook ${hookName} failed:`, error);
      // Continue with other plugins
    }
  }

  return result;
}
```

15.10.3 WebSocket Reconnect Storms

Problem: Exponential backoff too aggressive or too slow

Solution: Use capped exponential backoff

```
attemptReconnect() {
  this.reconnectAttempts++;

  // Min 1s, max 30s, exponential between
  const delay = Math.min(
    1000 * Math.pow(2, this.reconnectAttempts),
    30000
  );

  setTimeout(() => this.connect(), delay);
}
```

15.10.4 Micro-Frontend Memory Leaks

Problem: Modules not properly cleaned up on unload

Solution: Enforce destroy lifecycle

```
unloadModule(name) {
  const module = this.modules.get(name);

  if (!module) return;

  // Call destroy if exists
  if (module.destroy && typeof module.destroy === 'function') {
    try {
      module.destroy();
    } catch (error) {
      console.error(`Module ${name} destroy failed:`, error);
    }
  }

  // Remove DOM elements
  const container = this.querySelector(`[data-module="${name}"]`);
  if (container) container.remove();

  this.modules.delete(name);
}
```

15.10.5 Middleware Performance

Problem: Too many middleware slow down message delivery

Solution: Profile and optimize critical path

```
// Add timing middleware in dev mode
if (import.meta.env.DEV) {
  registry.register('timing', {
    priority: 1000, // Run first
    middleware: async (context) => {
      const start = performance.now();
      context = await next(context);
      const duration = performance.now() - start;

      if (duration > 16) { // Longer than 1 frame
        console.warn(`Slow middleware for ${context.type}:
        ${duration}ms`);
      }

      return context;
    }
  });
}
```

See Also: *Learning LARC* Chapter 7 for hands-on advanced pattern tutorials.

16 Deployment and Production

Quick reference for deploying LARC applications to production. For detailed tutorials, see *Learning LARC* Chapter 16.

16.1 Overview

LARC applications deploy as static files with optional build optimization. No complex pipelines required—serve vanilla JavaScript directly or optimize with minimal tooling.

Key Concepts: - No-build deployment option - Optional esbuild optimization - CDN deployment strategies - Caching policies - Performance monitoring - Production debugging tools

16.2 Quick Example

```
// Minimal esbuild configuration
import * as esbuild from 'esbuild';

await esbuild.build({
  entryPoints: ['src/app.js'],
  bundle: true,
  minify: true,
  sourcemap: true,
  target: ['es2020'],
  outfile: 'dist/app.js',
  format: 'esm'
});
```

Deploy `dist/` folder to any static host (Cloudflare Pages, Netlify, Vercel).

16.3 Build Options

Approach	Use Case	Complexity
No Build	Small apps, rapid prototyping	None
esbuild	Production optimization	Minimal
Code Splitting	Large apps with routing	Low
TypeScript	Type safety	Medium

16.3.1 Build Configuration Example

```
{
  "scripts": {
    "build": "node build.js",
    "dev": "node build.js --watch"
  }
}
```

16.4 CDN Deployment

16.4.1 Platform Commands

Platform	Command	Config File
Cloudflare Pages	`wrangler pages publish dist`	`_headers`
Netlify	`netlify deploy --dir=dist --prod`	`netlify.toml`
Vercel	`vercel --prod`	`vercel.json`

16.4.2 Cache Headers

```
# _headers (Cloudflare)
/*.js
  Cache-Control: public, max-age=31536000, immutable

/index.html
  Cache-Control: no-cache

/service-worker.js
  Cache-Control: no-cache
```

16.5 Common Patterns

16.5.1 Pattern 1: Service Worker Caching

Offline support and faster loads.

```
// service-worker.js
const CACHE_NAME = 'larc-app-v1';
const URLS_TO_CACHE = ['/', '/index.html', '/app.js',
        '/styles.css'];

self.addEventListener('install', (event) => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(cache => cache.addAll(URLS_TO_CACHE))
  );
});

self.addEventListener('fetch', (event) => {
  event.respondWith(
    caches.match(event.request)
      .then(response => response || fetch(event.request))
  );
});

// Register in app
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('/service-worker.js');
}
```

16.5.2 Pattern 2: Performance Monitoring

Track Core Web Vitals.

```
class PerformanceMonitor extends HTMLElement {
  connectedCallback() {
    // Largest Contentful Paint
    new PerformanceObserver((list) => {
      const entries = list.getEntries();
      const lcp = entries[entries.length - 1];
      this.sendMetric('lcp', lcp.renderTime || lcp.loadTime);
    }).observe({ entryTypes: ['largest-contentful-paint'] });

    // First Input Delay
    new PerformanceObserver((list) => {
      for (const entry of list.getEntries()) {
        const fid = entry.processingStart - entry.startTime;
        this.sendMetric('fid', fid);
      }
    }).observe({ entryTypes: ['first-input'] });

    // Cumulative Layout Shift
    let clsScore = 0;
    new PerformanceObserver((list) => {
      for (const entry of list.getEntries()) {
        if (!entry.hadRecentInput) clsScore += entry.value;
      }
      this.sendMetric('cls', clsScore);
    }).observe({ entryTypes: ['layout-shift'] });
  }

  sendMetric(name, value) {
    navigator.sendBeacon('/api/metrics', JSON.stringify({
      metric: name,
      value,
      url: location.pathname,
```

```
        timestamp: Date.now()
      }));
    }
  }

  customElements.define('perf-monitor', PerformanceMonitor);
```

16.5.3 Pattern 3: Error Tracking

Production error monitoring.

```
class ErrorTracker extends HTMLElement {
  connectedCallback() {
    // Global errors
    window.addEventListener('error', (e) => {
      this.trackError({
        message: e.message,
        stack: e.error?.stack,
        source: e.filename,
        line: e.lineno
      });
    });

    // Unhandled promise rejections
    window.addEventListener('unhandledrejection', (e) => {
      this.trackError({
        message: e.reason?.message || 'Promise rejected',
        stack: e.reason?.stack
      });
    });
  }

  trackError(error) {
    navigator.sendBeacon('/api/errors', JSON.stringify({
      ...error,
      timestamp: new Date().toISOString(),
      url: location.href,
      userAgent: navigator.userAgent
    }));
  }
}

customElements.define('error-tracker', ErrorTracker);
```

16.5.4 Pattern 4: Feature Flags

Control features without redeployment.

```
class FeatureFlags extends HTMLElement {
  async connectedCallback() {
    const bus = document.querySelector('pan-bus');

    // Load flags from server
    const response = await fetch('/api/feature-flags');
    this.flags = await response.json();

    bus.publish('flags.loaded', this.flags);

    // Check flags
    bus.subscribe('flags.check', (msg) => {
      const { flag, defaultValue } = msg.data;
      const enabled = this.flags[flag] ?? defaultValue;
      bus.publish('flags.result', { flag, enabled });
    });
  }
}

// Usage in components
bus.publish('flags.check', { flag: 'new-feature', defaultValue: false
        });
bus.subscribe('flags.result', (msg) => {
  if (msg.data.flag === 'new-feature' && msg.data.enabled) {
    // Show new feature
  }
});
```

16.6 Caching Strategies

Resource Type	Cache Duration	Strategy
HTML	No cache	`no-cache, must-revalidate`
JavaScript/ CSS	1 year	`public, max-age=31536000, immutable`
Images	30 days	`public, max-age=2592000`
API Responses	5 minutes	Client-side cache with invalidation

16.6.1 API Response Caching

```
class CachedAPIClient extends HTMLElement {
  constructor() {
    super();
    this.cache = new Map();
    this.cacheDuration = 5 * 60 * 1000; // 5 minutes
  }

  async fetch(method, endpoint, data) {
    const cacheKey = `${method}:${endpoint}`;

    // Check cache for GET
    if (method === 'GET') {
      const cached = this.cache.get(cacheKey);
      if (cached && Date.now() < cached.expiresAt) {
        return cached.data;
      }
    }

    // Fetch from API
    const response = await fetch(endpoint, {
      method,
      headers: { 'Content-Type': 'application/json' },
      body: data ? JSON.stringify(data) : undefined
    });

    const result = await response.json();

    // Cache GET responses
    if (method === 'GET') {
      this.cache.set(cacheKey, {
        data: result,
        expiresAt: Date.now() + this.cacheDuration
```

```
      });
    } else {
      // Invalidate cache on mutations
      this.cache.clear();
    }

    return result;
  }
}
```

16.7 Performance Metrics

16.7.1 Core Web Vitals Targets

Metric	Good	Needs Improvement	Poor
LCP (Largest Contentful Paint)	≤2.5s	2.5-4s	>4s
FID (First Input Delay)	≤100ms	100-300ms	>300ms
CLS (Cumulative Layout Shift)	≤0.1	0.1-0.25	>0.25
TTFB (Time to First Byte)	≤800ms	800-1800ms	>1800ms

16.7.2 Custom Performance Marks

```
// Track specific operations
performance.mark('load-start');
await fetchData();
performance.mark('load-end');
performance.measure('data-load', 'load-start', 'load-end');

const measurement = performance.getEntriesByName('data-load')[0];
console.log(`Data load: ${measurement.duration}ms`);
```

16.8 Complete Example

Production-ready deployment with monitoring.

```
// build.js - Build script with optimization
import * as esbuild from 'esbuild';
import { statSync } from 'fs';

const result = await esbuild.build({
  entryPoints: ['src/app.js'],
  bundle: true,
  minify: true,
  sourcemap: 'external',
  target: ['es2020'],
  outfile: 'dist/app.js',
  metafile: true
});

// Check bundle size
const stats = statSync('dist/app.js');
const sizeKB = (stats.size / 1024).toFixed(2);
console.log(`Bundle: ${sizeKB} KB`);

if (parseFloat(sizeKB) > 500) {
  throw new Error(`Bundle too large: ${sizeKB} KB`);
}

// index.html - Production HTML
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
        scale=1.0">
  <title>LARC App</title>
  <script type="module" src="/app.js"></script>
```

```
</head>
<body>
  <!-- Core components -->
  <pan-bus debug="false"></pan-bus>
  <error-tracker></error-tracker>
  <perf-monitor></perf-monitor>
  <feature-flags></feature-flags>

  <!-- App root -->
  <div id="app"></div>
</body>
</html>

// netlify.toml - Netlify configuration
[build]
  publish = "dist"
  command = "npm run build"

[[headers]]
  for = "/*.js"
  [headers.values]
    Cache-Control = "public, max-age=31536000, immutable"

[[headers]]
  for = "/index.html"
  [headers.values]
    Cache-Control = "no-cache"

[[redirects]]
  from = "/*"
  to = "/index.html"
  status = 200
```

```
// app.js - Application entry
import './core/pan-bus.mjs';
import './components/error-tracker.js';
import './components/perf-monitor.js';
import './components/feature-flags.js';

const bus = document.querySelector('pan-bus');

// Wait for initialization
bus.addEventListener('pan:sys.ready', () => {
  console.log('LARC app initialized');

  // Register service worker
  if ('serviceWorker' in navigator) {
    navigator.serviceWorker.register('/service-worker.js')
      .then(() => console.log('Service worker registered'))
      .catch(err => console.error('SW registration failed:', err));
  }

  // Load main app
  import('./main.js').then(module => {
    module.init(bus);
  });
});
```

16.9 Deployment Checklist

Pre-deployment verification:

- [ ] Tests pass
- [ ] Bundle minified and under size limit
- [ ] Source maps generated (protected in production)

- [ ] Cache headers configured
- [ ] Service worker registered (if using)
- [ ] Error tracking enabled
- [ ] Performance monitoring active
- [ ] Feature flags configured
- [ ] Environment variables set
- [ ] SSL certificate valid
- [ ] CDN configured
- [ ] Monitoring alerts set up
- [ ] Rollback plan documented

16.10 Component Reference

- Performance monitoring: Use native PerformanceObserver API
- Error tracking: Integrate with Sentry, Rollbar, or custom backend
- Feature flags: Server-side control with client-side caching

16.11 Cross-References

- **Tutorial**: *Learning LARC* Chapter 16 (Deployment)
- **Performance**: Chapter 12 (Performance Optimization)
- **Testing**: Chapter 13 (Testing Strategies)
- **Error Handling**: Chapter 14 (Error Handling and Debugging)
- **Configuration**: Appendix C (Build and Deploy Configuration)

16.12 Common Issues

16.12.1 Service Worker Not Updating

Problem: Users see stale cached content

Solution: Implement cache versioning and force update

```
const CACHE_NAME = 'larc-app-v2'; // Increment version

self.addEventListener('activate', (event) => {
  event.waitUntil(
    caches.keys().then(keys => {
      return Promise.all(
        keys.map(key => {
          if (key !== CACHE_NAME) {
            return caches.delete(key);
          }
        })
      );
    })
  );
});
```

16.12.2 Source Maps Exposed to Public

Problem: Source maps leak implementation details

Solution: Serve source maps only to authenticated users

```
// Edge function
export default {
  async fetch(request) {
    const url = new URL(request.url);

    if (url.pathname.endsWith('.map')) {
      const token = request.headers.get('Authorization');
      if (!isValidDevToken(token)) {
        return new Response('Unauthorized', { status: 401 });
      }
    }

    return fetch(request);
  }
};
```

16.12.3 Cache Invalidation After Deploy

Problem: Users load old cached assets with new HTML

Solution: Use cache busting with content hashes

```
// build.js - Add hash to filenames
import { createHash } from 'crypto';

const content = readFileSync('dist/app.js');
const hash = createHash('sha256')
  .update(content)
  .digest('hex')
  .slice(0, 8);

renameSync('dist/app.js', `dist/app.${hash}.js`);

// Update HTML references
let html = readFileSync('src/index.html', 'utf-8');
html = html.replace('app.js', `app.${hash}.js`);
writeFileSync('dist/index.html', html);
```

16.12.4 Performance Metrics Not Reporting

Problem: sendBeacon blocked by CORS or content blockers

Solution: Use fallback and proper CORS headers

```
sendMetric(name, value) {
  const payload = JSON.stringify({ metric: name, value });

  // Try sendBeacon first
  const sent = navigator.sendBeacon('/api/metrics', payload);

  // Fallback to fetch
  if (!sent) {
    fetch('/api/metrics', {
      method: 'POST',
      body: payload,
      keepalive: true
    }).catch(err => console.error('Metric send failed:', err));
  }
}
```

16.12.5 Bundle Size Exceeds Limits

Problem: Build fails due to large bundle

Solution: Implement code splitting and lazy loading

```
// Router with lazy loading
async loadRoute(route) {
  const module = await import(`./routes/${route}.js`);
  return new module.default();
}

// esbuild with splitting
await esbuild.build({
  entryPoints: ['src/app.js', 'src/routes/*.js'],
  bundle: true,
  splitting: true,
  format: 'esm',
  outdir: 'dist'
});
```

16.12.6 Update Notifications Not Showing

Problem: Users miss version updates

Solution: Implement version check with notification

```
class UpdateChecker extends HTMLElement {
  async connectedCallback() {
    const currentVersion = '1.2.3';

    const response = await fetch('/version.json', { cache: 'no-
        cache' });
    const { version } = await response.json();

    if (version !== currentVersion) {
      this.innerHTML = `
        <div class="update-banner">
          New version available!
          <button onclick="location.reload()">Refresh</button>
        </div>
      `;
    }
  }
}

customElements.define('update-checker', UpdateChecker);
```

See Also: *Learning LARC* Chapter 16 for step-by-step deployment tutorials.

17 Core Components Reference

API documentation for LARC's core components. For tutorials, see *Learning LARC* Chapters 4-5.

17.1 pan-bus

Purpose: Message bus for decoupled component communication via pub/sub **Import**: `<script type="module" src="/core/pan-bus.mjs"></script>`

17.1.1 Quick Example

```
<pan-bus debug="true"></pan-bus>

<script type="module">
const bus = document.querySelector('pan-bus');

// Subscribe
bus.subscribe('user.login', (msg) => {
  console.log('User:', msg.data.name);
});

// Publish
bus.publish('user.login', { name: 'Alice' });
</script>
```

17.1.2 Attributes

Attribute	Type	Default	Description
`max-retained`	Integer	1000	Max retained messages (LRU eviction)
`max-message-size`	Integer	1048576 (1MB)	Max message size (bytes)
`cleanup-interval`	Integer	30000 (30s)	Cleanup interval (ms)
`rate-limit`	Integer	1000	Max messages/client/second
`allow-global-wildcard`	Boolean	true	Allow `*` wildcard subscriptions
`debug`	Boolean	false	Enable console logging
`enable-routing`	Boolean	false	Enable declarative routing
`enable-tracing`	Boolean	false	Enable message tracing

17.1.3 Methods

Method	Parameters	Returns	Description
`publish(topic, data, options)`	topic: String, data: Any, options?: Object	undefined	Publish message
`subscribe(topics, handler)`	topics: String|Array, handler: Function	Function	Subscribe (returns unsubscribe fn)
`PanBusEnhanced.matches(topic, pattern)`	topic: String, pattern: String	Boolean	Test if topic matches pattern

Usage:

```
const bus = document.querySelector('pan-bus');

// Publish
bus.publish('user.login', { userId: 123 });
bus.publish('config', { theme: 'dark' }, { retain: true });

// Subscribe
const unsub = bus.subscribe('user.*', (msg) => {
  console.log(msg.topic, msg.data);
});

// Unsubscribe
unsub();
```

17.1.4 Events

Incoming (components dispatch these):

- `pan:hello` - Register client: `{ id, caps }`
- `pan:subscribe` - Subscribe: `{ topics, clientId, options }`
- `pan:unsubscribe` - Unsubscribe: `{ topics, clientId }`
- `pan:publish` - Publish: `{ topic, data, retain, clientId }`
- `pan:request` - Request (like publish): `{ topic, data, requestId }`
- `pan:sys.stats` - Get statistics
- `pan:sys.clear-retained` - Clear retained: `{ pattern? }`

Outgoing (bus dispatches these):

- `pan:sys.ready` - Bus ready: `{ enhanced, routing, tracing, config }`
- `pan:sys.error` - Error: `{ code, message, details }`
- `pan:deliver` - Deliver message: `{ topic, data, id, ts, ...extras }`

17.1.5 Complete Example

```
// Publisher component
class LoginForm extends HTMLElement {
  connectedCallback() {
    this.innerHTML = '<button id="login">Login</button>';

    this.querySelector('#login').addEventListener('click', () => {
      const bus = document.querySelector('pan-bus');
      bus.publish('user.login', {
        userId: '123',
        name: 'Alice',
        timestamp: Date.now()
      });
    });
  }
}

// Subscriber component
class Dashboard extends HTMLElement {
  connectedCallback() {
    const bus = document.querySelector('pan-bus');

    this.unsubscribe = bus.subscribe('user.login', (msg) => {
      this.innerHTML = `<h1>Welcome, ${msg.data.name}!</h1>`;
    });
  }

  disconnectedCallback() {
    if (this.unsubscribe) this.unsubscribe();
  }
}

customElements.define('login-form', LoginForm);
```

```
customElements.define('dash-board', Dashboard);
```

17.1.6 Common Issues

Messages not delivered: Wait for `pan:sys.ready` event or check `window.__panReady` **Memory leaks**: Always unsubscribe in `disconnectedCallback()` **Rate limiting**: Increase `rate-limit` attribute for high-frequency apps **JSON errors**: Payloads must be JSON-serializable (no functions, DOM nodes, circular refs)

17.1.7 Errors

The pan-bus component dispatches `pan:sys.error` events for error conditions:

Error Code	Cause	Resolution
`RATE_LIMIT`	Client exceeds rate limit	Increase `rate-limit` attribute or reduce publish frequency
`MESSAGE_TOO_LARGE`	Message exceeds max size	Increase `max-message-size` or reduce payload size
`INVALID_TOPIC`	Topic name is invalid	Use alphanumeric + dots (e.g., `user.login`)
`SUBSCRIPTION_FAILED`	Handler threw exception	Fix handler code, check console for details
`RETAINED_OVERFLOW`	Too many retained messages	Increase `max-retained` or clean up old messages
`WILDCARD_DISABLED`	`*` wildcard used when disabled	Set `allow-global-wildcard="true"`

Error handling example:

```
const bus = document.querySelector('pan-bus');

// Listen for errors
bus.addEventListener('pan:sys.error', (e) => {
  const { code, message, details } = e.detail;
  console.error(`PAN Bus Error [${code}]:`, message, details);

  // Handle specific errors
  if (code === 'RATE_LIMIT') {
    // Slow down publishing
    enableThrottling();
  } else if (code === 'MESSAGE_TOO_LARGE') {
    // Split large payloads
    splitPayload(details.payload);
  }
});
```

Exceptions thrown:

- `TypeError` : Invalid parameters (e.g., non-string topic, non-function handler)
- `RangeError` : Attribute values out of bounds (e.g., negative rate-limit)

17.2 pan-theme-provider

Purpose: Manages app theme with system preference detection **Import**: `<script type="module" src="/ui/pan-theme-provider.mjs"></script>`

17.2.1 Quick Example

```
<pan-theme-provider theme="auto"></pan-theme-provider>

<style>
  :root[data-theme="light"] { --bg: #fff; --text: #000; }
  :root[data-theme="dark"] { --bg: #000; --text: #fff; }
  body { background: var(--bg); color: var(--text); }
</style>
```

17.2.2 Attributes

Attribute	Type	Default	Description
`theme`	String	“auto”	Theme mode: “light”, “dark”, or “auto”

17.2.3 Methods

Method	Returns	Description
`setTheme(theme)`	undefined	Set theme: “light”, “dark”, or “auto”
`getTheme()`	String	Get theme mode
`getEffectiveTheme()`	String	Get actual theme (“light” or “dark”)
`getSystemTheme()`	String	Get system preference

Usage:

```
const provider = document.querySelector('pan-theme-provider');
provider.setTheme('dark');
console.log(provider.getEffectiveTheme());  // "dark"
```

17.2.4 Events

- **DOM**: `theme-change` - Detail: `{ theme, effective }`
- **PAN**: `theme.changed` - Data: `{ theme, effective }`
- **PAN**: `theme.system-changed` - Data: `{ theme }`

17.2.5 Complete Example

```
class ThemedApp extends HTMLElement {
  connectedCallback() {
    const bus = document.querySelector('pan-bus');

    // Subscribe to theme changes
    this.unsubscribe = bus.subscribe('theme.changed', (msg) => {
      this.applyTheme(msg.data.effective);
    });

    // Get initial theme
    const provider = document.querySelector('pan-theme-provider');
    if (provider) {
      this.applyTheme(provider.getEffectiveTheme());
    }

    // Persist preference
    bus.subscribe('theme.changed', (msg) => {
      localStorage.setItem('theme', msg.data.theme);
    });
  }

  applyTheme(theme) {
    this.className = `app theme-${theme}`;
  }

  disconnectedCallback() {
    if (this.unsubscribe) this.unsubscribe();
  }
}

customElements.define('themed-app', ThemedApp);
```

17.2.6 Common Issues

Theme not applying: Define CSS variables for both `[data-theme="light"]` and `[data-theme="dark"]` **Flash on load**: Set theme in inline `<script>` before loading components **Not updating**: Subscribe to `theme.changed` in `connectedCallback()`

17.2.7 Errors

Error Condition	Cause	Resolution
Invalid theme value	`setTheme()` called with invalid value	Use “light”, “dark”, or “auto” only
`matchMedia` not supported	Old browser	Provide polyfill or default to “light”
localStorage quota	Storage full	Clear old data or use memory-only mode

Error handling example:

```
const provider = document.querySelector('pan-theme-provider');

try {
  provider.setTheme('custom'); // Invalid!
} catch (e) {
  console.error('Theme error:', e.message);
  // Fallback to auto
  provider.setTheme('auto');
}
```

Exceptions thrown:

- `TypeError` : Invalid theme value passed to `setTheme()`
- `DOMException` : localStorage access denied (privacy mode)

17.3 pan-theme-toggle

Purpose: UI control for theme switching **Import**: `<script type="module" src="/ui/pan-theme-toggle.mjs"></script>`

17.3.1 Quick Example

```
<pan-theme-toggle></pan-theme-toggle>
<pan-theme-toggle variant="button" label="Theme"></pan-theme-toggle>
<pan-theme-toggle variant="dropdown"></pan-theme-toggle>
```

17.3.2 Attributes

Attribute	Type	Default	Description
`label`	String	“”	Text label (button variant only)
`variant`	String	“icon”	Style: “icon”, “button”, or “dropdown”

17.3.3 Complete Example

```
<nav class="toolbar">
  <h1>My App</h1>
  <div class="actions">
    <button>Settings</button>
    <pan-theme-toggle></pan-theme-toggle>
  </div>
</nav>

<style>
  .toolbar {
    display: flex;
    justify-content: space-between;
    padding: 1rem;
    background: var(--surface);
  }

  .actions {
    display: flex;
    gap: 0.5rem;
    align-items: center;
  }
</style>
```

17.3.4 Common Issues

Not working: Ensure `pan-theme-provider` exists **Wrong icon**: Component subscribes on connect; add after `pan:sys.ready` **Dropdown positioning**: Wrap in `position: relative` container

17.3.5 Errors

Error Condition	Cause	Resolution
Provider not found	`pan-theme-provider` missing	Add `<pan-theme-provider>` to page
Invalid variant	Unsupported `variant` attribute	Use “icon”, “button”, or “dropdown”
Event not dispatched	pan-bus not ready	Wait for `pan:sys.ready` or add toggle after bus loads

Error handling example:

```
// Ensure provider exists
window.addEventListener('DOMContentLoaded', () => {
  const toggle = document.querySelector('pan-theme-toggle');
  const provider = document.querySelector('pan-theme-provider');

  if (!provider) {
    console.warn('pan-theme-toggle requires pan-theme-provider');
    // Add provider dynamically
    const newProvider = document.createElement('pan-theme-
        provider');
    document.body.prepend(newProvider);
  }
});
```

Exceptions thrown: None (component fails gracefully without provider)

17.4 pan-routes

Purpose: Runtime-configurable message routing **Import**: Enabled via `<pan-bus enable-routing="true"></pan-bus>` **Access**: `window.pan.routes` or `bus.routingManager`

17.4.1 Quick Example

```
const routes = window.pan.routes;

// Add route
routes.add({
  name: 'Login redirect',
  match: { type: 'user.login.success' },
  actions: [
    {
      type: 'EMIT',
      message: { topic: 'ui.navigate', data: { to: '/dashboard' } }
    }
  ]
});
```

17.4.2 Methods

Method	Parameters	Returns	Description
`add(route)`	route: Object	Object	Add route (returns route with ID)
`update(id, patch)`	id: String, patch: Object	Object	Update route
`remove(id)`	id: String	Boolean	Remove route
`enable(id)` / `disable(id)`	id: String	-	Enable/disable route
`get(id)`	id: String	Object?	Get route by ID
`list(filter?)`	filter?: Object	Array	List routes
`clear()`	-	-	Remove all routes
`registerTransformFn(id, fn)`	id: String, fn: Function	-	Register transform
`registerHandler(id, fn)`	id: String, fn: Function	-	Register handler
`getStats()`	-	Object	Get statistics
`resetStats()`	-	-	Reset statistics
`setEnabled(bool)`	enabled: Boolean	-	Enable/disable routing
`onRoutesChanged(fn)`	fn: Function	Function	Subscribe to

Method	Parameters	Returns	Description
			changes (returns unsub)
`onError(fn)`	fn: Function	Function	Subscribe to errors (returns unsub)

17.4.3 Route Configuration

```
{
  id: String,           // Optional (auto-generated)
  name: String,         // Required
  enabled: Boolean,     // Default: true
  order: Number,        // Default: 0 (execution order)
  match: {              // Match criteria
    type: String|Array,     // Message type(s)
    topic: String|Array,    // Topic pattern(s)
    source: String|Array,   // Source client(s)
    tagsAny: Array,         // Has any tag
    tagsAll: Array,         // Has all tags
    where: Object           // Predicate (see below)
  },
  transform: Object,    // Optional transform
  actions: Array,       // Actions to perform
  meta: Object          // Optional metadata
}
```

17.4.4 Match Predicates

Operators: `eq`, `neq`, `gt`, `gte`, `lt`, `lte`, `in`, `regex`, `and`, `or`, `not`

```
// Simple
where: { op: 'gt', path: 'payload.temp', value: 30 }

// Regex
where: { op: 'regex', path: 'payload.email', value: '^[\\w-]+@' }

// Combined
where: {
  op: 'and',
  children: [
    { op: 'eq', path: 'payload.status', value: 'active' },
    { op: 'gt', path: 'payload.score', value: 75 }
  ]
}
```

17.4.5 Transform Operations

```
// Identity (no change)
transform: { op: 'identity' }

// Pick fields
transform: {
  op: 'pick',
  paths: ['payload.userId', 'meta.timestamp']
}

// Map with function (register first)
routes.registerTransformFn('double', (x) => x * 2);
transform: { op: 'map', path: 'payload.value', fnId: 'double' }

// Custom (register first)
routes.registerTransformFn('custom', (msg) => ({ ...msg, payload: {}
        }));
transform: { op: 'custom', fnId: 'custom' }
```

17.4.6 Actions

```
// EMIT - Publish new message
{
  type: 'EMIT',
  message: { topic: 'new.topic', data: {} },
  inherit: ['payload', 'meta']
}

// FORWARD - Forward to different topic
{
  type: 'FORWARD',
  topic: 'new.topic',
  typeOverride: 'new.type'  // Optional
}

// LOG - Console log
{
  type: 'LOG',
  level: 'info',  // 'log', 'info', 'warn', 'error'
  template: 'User {{payload.id}} action'
}

// CALL - Call registered handler
{
  type: 'CALL',
  handlerId: 'myHandler'
}
```

17.4.7 Complete Example

```
const routes = window.pan.routes;

// Register handlers
routes.registerHandler('notifyAdmin', async (msg) => {
  await fetch('/api/admin/notify', {
    method: 'POST',
    body: JSON.stringify(msg)
  });
});

routes.registerTransformFn('sanitize', (email) => {
  return email.toLowerCase().trim();
});

// Workflow: Registration -> Validation -> Welcome
routes.add({
  name: 'User registration workflow',
  order: 0,
  match: { type: 'user.register' },
  transform: {
    op: 'map',
    path: 'payload.email',
    fnId: 'sanitize'
  },
  actions: [
    {
      type: 'EMIT',
      message: { topic: 'email.validate' },
      inherit: ['payload']
    }
  ]
});
```

```
routes.add({
  name: 'Email validated -> Welcome',
  order: 10,
  match: { type: 'email.validated' },
  actions: [
    {
      type: 'EMIT',
      message: {
        topic: 'email.send',
        data: { template: 'welcome' }
      },
      inherit: ['payload']
    }
  ]
});

// Log high-value transactions
routes.add({
  name: 'High-value transaction alert',
  match: {
    type: 'transaction.complete',
    where: { op: 'gt', path: 'payload.amount', value: 1000 }
  },
  actions: [
    {
      type: 'LOG',
      level: 'warn',
      template: 'High-value: ${{payload.amount}}'
    },
    {
      type: 'CALL',
      handlerId: 'notifyAdmin'
```

```
    }
  ]
});

// Filter invalid sensor data
routes.add({
  name: 'Filter sensor readings',
  match: {
    type: 'sensor.reading',
    where: {
      op: 'and',
      children: [
        { op: 'gte', path: 'payload.temp', value: -40 },
        { op: 'lte', path: 'payload.temp', value: 85 }
      ]
    }
  },
  transform: {
    op: 'pick',
    paths: ['payload.temp', 'meta.sensorId']
  },
  actions: [
    { type: 'FORWARD', topic: 'sensor.valid' }
  ]
});

// Get stats
console.log(routes.getStats());
// {
//   routesEvaluated: 150,
//   routesMatched: 45,
//   actionsExecuted: 67,
//   errors: 0,
```

```
//   routeCount: 4,
//   enabledRouteCount: 4
// }
```

17.4.8 Common Issues

Routes not firing: Enable routing with `<pan-bus enable-routing="true">` **Wrong matches**: Enable debug mode: `<pan-bus debug="true">` **Transform not found**: Register functions before adding routes **Wrong order**: Set explicit `order` property (lower executes first)

17.4.9 Errors

Subscribe to errors using `onError()`:

Error Type	Cause	Resolution
`INVALID_ROUTE`	Missing required fields (`name` , `match` , or `actions`)	Provide all required fields in route config
`ROUTE_NOT_FOUND`	`update()` or `remove()` with invalid ID	Check route exists with `get(id)` first
`PREDICATE_ERROR`	Invalid `where` operator or structure	Use supported operators: `eq` , `gt` , `lt` , `in` , `regex` , `and` , `or` , `not`
`TRANSFORM_ERROR`	Transform function threw exception	Fix transform function or handle errors within it
`TRANSFORM_NOT_FOUND`	`fnId` not registered	Call `registerTransformFn(id, fn)` before using
`ACTION_ERROR`	Action execution failed	Check action configuration and registered handlers
`HANDLER_NOT_FOUND`	`CALL` action with unregistered handler	Call `registerHandler(id, fn)` before using

Error handling example:

```
const routes = window.pan.routes;

// Subscribe to errors
const unsubscribe = routes.onError((error) => {
  console.error('Route error:', error);

  // Handle specific errors
  switch (error.code) {
    case 'TRANSFORM_ERROR':
      // Disable problematic route
      routes.disable(error.routeId);
      notifyAdmin('Route failed', error);
      break;

    case 'HANDLER_NOT_FOUND':
      // Register missing handler
      routes.registerHandler(error.handlerId, fallbackHandler);
      break;
  }
});

// Validate route before adding
try {
  routes.add({
    name: 'My Route',
    match: { type: 'test' },
    actions: [{ type: 'LOG', level: 'info', template: 'Test' }]
  });
} catch (e) {
  console.error('Failed to add route:', e.message);
}
```

Exceptions thrown:

- `Error`: Invalid route configuration (missing required fields)
- `TypeError`: Invalid parameters (e.g., non-function handler)
- `RangeError`: Invalid `order` value

17.5 Summary

This chapter documented LARC's four core components:

- **pan-bus**: Message bus infrastructure for pub/sub communication
- **pan-theme-provider**: Centralized theme management
- **pan-theme-toggle**: UI controls for theme switching
- **pan-routes**: Message routing for workflows and orchestration

These form the backbone of every LARC application.

See Also:

- Tutorial: *Learning LARC* Chapters 4-5
- State components: Chapter 18
- UI components: Chapter 19
- Message patterns: Appendix A
- Configuration: Appendix C

18 Data Components Reference

API documentation for LARC's data management components. For tutorials, see *Learning LARC* Chapter 6.

18.1 pan-store

Purpose: Reactive state management for shared application state
Import: `import { createStore, bind } from './pan-store.mjs';`

18.1.1 Quick Example

```
import { createStore } from './pan-store.mjs';

const store = createStore({ count: 0, theme: 'light' });

// Subscribe to changes
store.subscribe(({ detail }) => {
  console.log(`${detail.key} changed to ${detail.value}`);
});

// Update state
store.state.count++; // Triggers subscriber
```

18.1.2 API

18.1.2.1 createStore(initial)

Creates reactive store with optional initial state.

Parameters: - `initial` (Object, optional): Initial state

Returns: Store instance

Properties: - `state` (Proxy): Reactive state object

18.1.2.2 Store Methods

Method	Parameters	Returns	Description
`subscribe(callback)`	callback: Function	Function	Subscribe to changes (returns unsubscribe fn)
`set(key, value)`	key: String, value: Any	-	Set single property
`patch(object)`	object: Object	-	Merge multiple properties
`update(fn)`	fn: Function	-	Update using function
`select(path)`	path: String	Any	Get nested value by dot-path
`derive(key, deps, computeFn)`	key: String, deps: Array, computeFn: Function	Function	Create computed property
`batch(fn)`	fn: Function	-	Batch multiple updates
`use(middleware)`	middleware: Function	Function	Add middleware (returns unsubscribe)
`snapshot()`	-	Object	Deep clone of current state

Method	Parameters	Returns	Description
`reset()`	-	-	Reset to initial values
`has(key)`	key: String	Boolean	Check if property exists
`delete(key)`	key: String	Boolean	Remove property
`keys()`	-	Array	Get all property names

Usage:

```
const store = createStore({ count: 0 });

// Direct access
store.state.count++; // Triggers updates

// Methods
store.set('theme', 'dark');
store.patch({ count: 5, theme: 'dark' });
store.update(state => { state.count += 1; return state; });

// Derived values
store.derive('doubled', ['count'], (count) => count * 2);
console.log(store.state.doubled); // 10

// Batching
store.batch(({ set }) => {
  set('loading', true);
  set('error', null);
}); // Single event

// Cleanup
const unsub = store.subscribe(handler);
unsub();
```

18.1.3 Events

- **state**: Emitted on state change
  Detail: `{ key, value, oldValue, state, batch?, changes?, deleted? }`
- **derived**: Emitted on derived value update
  Detail: `{ key, value, state }`

18.1.4 bind(element, store, mapping, options)

Two-way binding for form inputs.

Parameters: - `element` (HTMLElement): Container element - `store` (Store): Store instance - `mapping` (Object): CSS selectors to property names - `options` (Object, optional): `{ events: ['input', 'change'] }`

Returns: Unbind function

Usage:

```
const store = createStore({ username: '', email: '' });
const form = document.querySelector('#user-form');

const unbind = bind(form, store, {
  'input[name="username"]': 'username',
  'input[name="email"]': 'email'
});

// Input changes update store
// Store changes update inputs
```

18.1.5 Complete Example

```
import { createStore } from './pan-store.mjs';

// Shopping cart with derived totals
const cart = createStore({
  items: [
    { id: 1, name: 'Widget', price: 10, qty: 2 },
    { id: 2, name: 'Gadget', price: 25, qty: 1 }
  ],
  taxRate: 0.08
});

// Derive subtotal
cart.derive('subtotal', ['items'], (items) => {
  return items.reduce((sum, item) => sum + (item.price * item.qty),
        0);
});

// Derive tax and total
cart.derive('tax', ['subtotal', 'taxRate'], (sub, rate) => sub *
        rate);
cart.derive('total', ['subtotal', 'tax'], (sub, tax) => sub + tax);

// Add logging middleware
cart.use(({ key, value }) => {
  console.log(`State changed: ${key} = ${value}`);
});

// Subscribe to total changes
cart.subscribe(({ detail }) => {
  if (detail.key === 'total') {
    console.log(`Cart total: $${detail.value.toFixed(2)}`);
  }
});
```

```
// Access computed values
console.log(cart.state.subtotal); // 45
console.log(cart.state.tax);      // 3.6
console.log(cart.state.total);    // 48.6
```

18.1.6 Errors

pan-store emits errors through the `error` event for error conditions:

Error Condition	Cause	Resolution
Circular dependency	Derived value depends on itself	Review `derive()` dependency chains
Invalid path	`select()` with non-string path	Use valid dot-notation string (e.g., `'user.name'`)
Middleware error	Middleware function throws exception	Wrap middleware in try-catch or fix thrown error
Frozen state	Attempting to modify frozen object	Unfreeze object or create new reference
Type error	Non-object passed to `patch()`	Pass plain object to `patch()`

Error handling example:

```
const store = createStore({ count: 0 });

// Listen for errors
store.addEventListener('error', (e) => {
  const { error, operation, key } = e.detail;
  console.error(`Store error in ${operation}:`, error.message);

  // Handle specific errors
  if (error.message.includes('Circular')) {
    // Remove circular dependencies
    console.warn(`Circular dependency detected for key: ${key}`);
  } else if (operation === 'middleware') {
    // Middleware failed
    console.error('Middleware error:', error);
  }
});

// Safe derived value with error handling
try {
  store.derive('computed', ['dep1'], (dep) => {
    if (!dep) throw new Error('Invalid dependency');
    return dep * 2;
  });
} catch (err) {
  console.error('Failed to create derived value:', err);
}

// Safe middleware usage
store.use((state, changes) => {
  try {
    // Validate changes
    if (changes.price && changes.price < 0) {
```

```
      throw new Error('Price cannot be negative');
    }
    return changes;
  } catch (err) {
    console.error('Middleware validation failed:', err);
    return null; // Reject changes
  }
});
```

Exceptions thrown:

- `TypeError`: Invalid parameters (e.g., non-string path, non-function callback)
- `RangeError`: Invalid computed dependency (circular reference)
- `Error`: Middleware exceptions (propagated from user code)

18.1.7 Common Issues

Nested changes not detected

```
// Problem
store.state.user.name = 'Ada'; // No event

// Solution: reassign parent
store.state.user = { ...store.state.user, name: 'Ada' };
```

Performance with frequent updates

```
// Problem: 1000 individual events
for (let i = 0; i < 1000; i++) store.state.count = i;

// Solution: use batch()
store.batch(({ set }) => {
  for (let i = 0; i < 1000; i++) set('count', i);
});
```

Memory leaks

```
// Problem: no cleanup
class MyComponent extends HTMLElement {
  connectedCallback() {
    store.subscribe(this.handleChange); // Leaks
  }
}

// Solution: unsubscribe on disconnect
class MyComponent extends HTMLElement {
  connectedCallback() {
    this.unsub = store.subscribe(this.handleChange);
  }
  disconnectedCallback() {
    if (this.unsub) this.unsub();
  }
}
```

18.2 pan-idb

Purpose: IndexedDB integration via PAN message bus
Import: `<pan-idb>` custom element

18.2.1 Quick Example

```
<pan-idb
  database="myapp"
  store="documents"
  key-path="id"
  auto-increment
  indexes='[{"name":"byTitle","keyPath":"title"}]'>
</pan-idb>

<script type="module">
import { PanClient } from './pan-client.mjs';

const pc = new PanClient();

// Add document
pc.publish({
  topic: 'documents.idb.add',
  data: { item: { title: 'Report', content: '...' } }
});

// Listen for result
pc.subscribe('documents.idb.result', (msg) => {
  console.log('Saved with key:', msg.data.key);
});
</script>
```

18.2.2 Attributes

Attribute	Type	Default	Description
`database`	String	-	Database name (required)
`version`	Number	1	Database version
`store`	String	-	Object store name (required)
`key-path`	String	“id”	Primary key property
`auto-increment`	Boolean	false	Use auto-incrementing keys
`indexes`	JSON String	[]	Index configurations

Index format:

```
[
  {
    "name": "byTitle",
    "keyPath": "title",
    "unique": false,
    "multiEntry": false
  }
]
```

18.2.3 PAN Topics

All topics follow pattern `{store}.idb.{operation}`.

18.2.3.1 Subscribe (Commands)

Topic	Data	Description
`{store}.idb.get`	`{ key }`	Retrieve item by key
`{store}.idb.put`	`{ item }`	Insert or update item
`{store}.idb.add`	`{ item }`	Insert item (fails if exists)
`{store}.idb.delete`	`{ key }`	Delete item by key
`{store}.idb.clear`	`{}`	Delete all items
`{store}.idb.list`	`{ index?, range?, direction?, limit? }`	List items
`{store}.idb.query`	`{ index, value }`	Query by index
`{store}.idb.count`	`{ index? }`	Count items

18.2.3.2 Publish (Results)

Topic	Data	Description
`{store}.idb.ready`	`{ database, store }`	Database initialized
`{store}.idb.result`	`{ operation, success, ...data }`	Operation succeeded
`{store}.idb.error`	`{ operation, success: false, error }`	Operation failed

Result data by operation: - `get`: `{ item }` - `put` / `add`: `{ key }` - `list` / `query`: `{ items }` - `count`: `{ count }`

18.2.4 Methods

Direct JavaScript API (alternative to PAN topics):

Method	Parameters	Returns	Description
`get(key)`	key: Any	Promise<Object>	Retrieve item
`put(item)`	item: Object	Promise<Any>	Insert/update item
`add(item)`	item: Object	Promise<Any>	Insert item
`delete(key)`	key: Any	Promise<void>	Delete item
`clear()`	-	Promise<void>	Delete all
`list(options)`	options: Object	Promise<Array>	List items
`query(index, value)`	index: String, value: Any	Promise<Array>	Query by index
`count(index)`	index?: String	Promise<Number>	Count items

Usage:

```
const idb = document.querySelector('pan-idb');
await customElements.whenDefined('pan-idb');
await idb.initPromise;

// CRUD operations
const id = await idb.add({ title: 'Report', status: 'draft' });
const doc = await idb.get(id);
doc.status = 'published';
await idb.put(doc);

// Query
const drafts = await idb.query('byStatus', 'draft');
const recent = await idb.list({
  index: 'byCreated',
  direction: 'prev',
  limit: 5
});

// Count and delete
const total = await idb.count();
await idb.delete(id);
```

18.2.5 Complete Example

```
<!DOCTYPE html>
<html>
<head><title>Document Manager</title></head>
<body>
  <pan-idb
    database="docapp"
    store="documents"
    key-path="id"
    auto-increment
    indexes='[
      {"name":"byTitle","keyPath":"title"},
      {"name":"byCreated","keyPath":"created"}
    ]'>
  </pan-idb>

  <form id="doc-form">
    <input name="title" placeholder="Title" required>
    <textarea name="content" placeholder="Content"></textarea>
    <button type="submit">Save</button>
  </form>

  <ul id="doc-list"></ul>

  <script type="module">
    import { PanClient } from './pan-client.mjs';

    const pc = new PanClient();
    const form = document.getElementById('doc-form');
    const list = document.getElementById('doc-list');

    // Wait for ready
    pc.subscribe('documents.idb.ready', loadDocuments);
```

```
// Save document
form.addEventListener('submit', (e) => {
  e.preventDefault();
  const formData = new FormData(form);

  pc.publish({
    topic: 'documents.idb.add',
    data: {
      item: {
        title: formData.get('title'),
        content: formData.get('content'),
        created: Date.now()
      }
    }
  });

  form.reset();
});

// Load documents
function loadDocuments() {
  pc.publish({
    topic: 'documents.idb.list',
    data: { index: 'byCreated', direction: 'prev', limit: 20 }
  });
}

// Render results
pc.subscribe('documents.idb.result', (msg) => {
  if (msg.data.operation === 'add') loadDocuments();
  if (msg.data.operation === 'list')
    renderDocuments(msg.data.items);
```

```
    if (msg.data.operation === 'delete') loadDocuments();
  });

  function renderDocuments(docs) {
    list.innerHTML = docs.map(doc => `
      <li>
        <strong>${doc.title}</strong>
        <p>${doc.content}</p>
        <small>${new Date(doc.created).toLocaleString()}</small>
        <button onclick="deleteDoc(${doc.id})">Delete</button>
      </li>
    `).join('');
  }

  window.deleteDoc = (id) => {
    pc.publish({ topic: 'documents.idb.delete', data: { key: id }
      });
  };
</script>
</body>
</html>
```

18.2.6 Errors

pan-idb publishes error messages to `{storeName}.idb.error` topic:

Error Code	Cause	Resolution
`DB_OPEN_FAILED`	Cannot open IndexedDB	Check browser support, quota, or permissions
`STORE_NOT_FOUND`	Object store doesn't exist	Set correct `store` attribute or upgrade database version
`VERSION_ERROR`	Version downgrade attempted	Remove old database or increment version number
`QUOTA_EXCEEDED`	Storage limit reached	Clear old data or request persistent storage
`TRANSACTION_FAILED`	Transaction aborted	Check data validity, reduce transaction size
`INDEX_ERROR`	Index operation failed	Verify index exists and data matches index keyPath
`KEY_ERROR`	Invalid key provided	Use valid key type (number, string, date, array)
`DATA_ERROR`	Data cannot be cloned	Remove non-cloneable values (functions, DOM nodes)
`READ_ONLY_ERROR`	Write on readonly transaction	Use correct transaction mode
`NOT_SUPPORTED`	Browser	Provide fallback

Error Code	Cause	Resolution
	doesn’t support IndexedDB	(localStorage, memory store)

Error handling example:

```
const bus = document.querySelector('pan-bus');
const idb = document.querySelector('pan-idb');

// Subscribe to errors
bus.subscribe('documents.idb.error', (msg) => {
  const { code, error, operation } = msg.data;
  console.error(`IDB error [${code}]:`, error.message);

  // Handle specific errors
  switch (code) {
    case 'QUOTA_EXCEEDED':
      // Prompt user to clear data
      if (confirm('Storage full. Clear old data?')) {
        bus.publish({ topic: 'documents.idb.clear' });
      }
      break;

    case 'VERSION_ERROR':
      // Database needs upgrade
      console.warn('Please reload to upgrade database');
      location.reload();
      break;

    case 'NOT_SUPPORTED':
      // Fallback to alternative storage
      console.warn('IndexedDB not available, using localStorage');
      initLocalStorageFallback();
      break;

    case 'DB_OPEN_FAILED':
      // Retry with exponential backoff
      setTimeout(() => {
```

```
        idb.setAttribute('database', idb.getAttribute('database'));
      }, 1000 * Math.pow(2, retryCount++));
      break;

    default:
      // Generic error handling
      showErrorNotification(`Database error: ${error.message}`);
  }
});

// Graceful degradation
bus.subscribe('documents.idb.error', async (msg) => {
  if (msg.data.code === 'NOT_SUPPORTED' || msg.data.code ===
      'DB_OPEN_FAILED') {
    // Switch to memory-only mode
    window.inMemoryStore = new Map();
    console.warn('Using in-memory storage (data will not persist)');
  }
});

// Quota management
async function checkQuota() {
  if (navigator.storage && navigator.storage.estimate) {
    const estimate = await navigator.storage.estimate();
    const percentUsed = (estimate.usage / estimate.quota) * 100;

    if (percentUsed > 90) {
      console.warn(`Storage ${percentUsed.toFixed(1)}% full`);
      // Trigger cleanup
      bus.publish({ topic: 'documents.idb.cleanup' });
    }
  }
}
```

Exceptions thrown:

- `DOMException`: IndexedDB-specific errors (InvalidStateError, ConstraintError, etc.)
- `TypeError`: Invalid parameters (e.g., non-cloneable data)
- `QuotaExceededError`: Storage limit reached
- `Error`: General database operation failures

Browser Compatibility Notes:

- IndexedDB supported in all modern browsers (Chrome 24+, Firefox 16+, Safari 10+)
- Private browsing may disable or limit IndexedDB
- Check support: `if ('indexedDB' in window)`

18.2.7 Common Issues

Database version conflicts: Different tabs with different versions

```
// Handle versionchange event
idb.db.addEventListener('versionchange', () => {
  idb.db.close();
  alert('Database upgraded. Please reload.');
});
```

Quota exceeded: Check available storage

```
if (navigator.storage && navigator.storage.estimate) {
  const estimate = await navigator.storage.estimate();
  const percent = (estimate.usage / estimate.quota) * 100;
  if (percent > 90) console.warn('Storage nearly full');
}
```

Index not working after schema changes: Increment version number

```
<!-- Change version="1" to version="2" -->
<pan-idb database="myapp" store="docs" version="2">
```

Transaction timeouts: Break bulk operations into batches

```
async function bulkInsert(items) {
  const BATCH_SIZE = 100;
  for (let i = 0; i < items.length; i += BATCH_SIZE) {
    const batch = items.slice(i, i + BATCH_SIZE);
    for (const item of batch) await idb.add(item);
    await new Promise(r => setTimeout(r, 0)); // Yield
  }
}
```

18.3 Summary

This chapter documented LARC's data management components:

- **pan-store**: Reactive state management with Proxy-based observation
- **pan-idb**: IndexedDB integration via PAN message bus

Use pan-store for reactive application state, pan-idb for persistent storage.

See Also:

- Tutorial: *Learning LARC* Chapter 6
- Message bus: Chapter 17
- UI components: Chapter 19
- State patterns: Appendix A

19 UI Components Reference

API documentation for LARC's UI components. For tutorials, see *Learning LARC* Chapter 10.

19.1 pan-files

Purpose: OPFS-backed file system browser with visual UI and programmatic API
Import: `<script type="module" src="/ui/pan-files.mjs"></script>`

19.1.1 Quick Example

```
<pan-files filter=".md,.txt"></pan-files>

<script type="module">
const files = document.querySelector('pan-files');

// Write file
await files.writeFile('/notes.txt', 'Hello, World!');

// Read file
const content = await files.readFile('/notes.txt');

// List files
const allFiles = await files.listFiles();
</script>
```

19.1.2 Attributes

Attribute	Type	Default	Description
`path`	String	"/"	Current directory path (root level only)
`filter`	String	""	Comma-separated file extensions to display
`show-hidden`	Boolean	false	Show files starting with dot

19.1.3 Methods

Method	Parameters	Returns	Description
`writeFile(path, content)`	path: String, content: String	Promise<void>	Write file (creates or overwrites)
`readFile(path)`	path: String	Promise<String>	Read file contents
`deleteFile(path)`	path: String	Promise<void>	Delete file
`listFiles()`	-	Promise<Array<FileInfo>>	Get all files in directory
`refresh()`	-	Promise<void>	Reload file list and update UI

FileInfo structure:

```
{
  name: 'example.txt',
  path: '/example.txt',
  isDirectory: false,
  size: 1024,
  entry: FileSystemHandle
}
```

19.1.4 PAN Events

19.1.4.1 Published

Topic	Data	Description
`file.selected`	`{ path, name, isDirectory }`	User clicked file
`file.created`	`{ path, name, isDirectory }`	File/folder created
`file.deleted`	`{ path }`	File deleted
`file.renamed`	`{ oldPath, newPath }`	File renamed
`file.content-loaded`	`{ path, content }`	Response to file.load

19.1.4.2 Subscribed

Topic	Data	Description
`file.save`	`{ path, content }`	Save file
`file.load`	`{ path }`	Load file (triggers file.content-loaded)
`file.delete`	`{ path }`	Delete file
`file.create`	`{ path, content? }`	Create file with optional content

19.1.5 Complete Example

```
<!DOCTYPE html>
<html>
<head>
  <script type="module">
    import './pan-bus.mjs';
    import './pan-files.mjs';

    customElements.whenDefined('pan-bus').then(() => {
      const bus = document.querySelector('pan-bus');
      const files = document.querySelector('pan-files');

      // Load file content when selected
      bus.subscribe('file.selected', (msg) => {
        bus.publish('file.load', { path: msg.data.path });
      });

      bus.subscribe('file.content-loaded', (msg) => {
        document.getElementById('preview').textContent =
        msg.data.content;
      });

      // Create quick note
      window.createNote = async () => {
        await files.writeFile(
          `/note-${Date.now()}.txt`,
          `Created at ${new Date().toISOString()}`
        );
        await files.refresh();
      };
    });
  </script>
  <style>
```

```
    body { display: grid; grid-template-columns: 300px 1fr; gap:
        1rem; padding: 1rem; }
    #preview { padding: 1rem; border: 1px solid #ccc; white-space:
        pre-wrap; font-family: monospace; }
  </style>
</head>
<body>
  <pan-bus></pan-bus>
  <pan-files filter=".txt,.md"></pan-files>
  <div>
    <button onclick="createNote()">Create Note</button>
    <pre id="preview">Select a file...</pre>
  </div>
</body>
</html>
```

19.1.6 Errors

pan-files dispatches custom `error` events and publishes to `file.error` topic:

Error Code	Cause	Resolution
`OPFS_NOT_SUPPORTED`	Browser doesn't support OPFS	Use Chrome 102+, Edge 102+, or provide fallback
`OPFS_INIT_FAILED`	Failed to initialize file system	Check HTTPS connection (or localhost)
`FILE_NOT_FOUND`	File doesn't exist	Verify path or catch error gracefully
`WRITE_FAILED`	Cannot write file	Check storage quota or permissions
`READ_FAILED`	Cannot read file	Verify file exists and is accessible
`DELETE_FAILED`	Cannot delete file	Check if file is locked or in use
`QUOTA_EXCEEDED`	Storage quota exceeded	Clear old files or request persistent storage
`INVALID_PATH`	Path contains invalid characters	Use valid path format (no `<>:"|?*`)

Error handling example:

```
const files = document.querySelector('pan-files');
const bus = document.querySelector('pan-bus');

// Listen for errors via DOM event
files.addEventListener('error', (e) => {
  const { code, message, path } = e.detail;
  console.error(`File error [${code}]:`, message);

  switch (code) {
    case 'OPFS_NOT_SUPPORTED':
      showFallbackUI();
      break;
    case 'QUOTA_EXCEEDED':
      promptUserToCleanup();
      break;
    case 'FILE_NOT_FOUND':
      console.warn(`File not found: ${path}`);
      break;
  }
});

// Or subscribe via PAN bus
bus.subscribe('file.error', (msg) => {
  const { code, message, path } = msg.data;
  displayErrorToUser(message);
});

// Safe file operations with try-catch
async function safeWriteFile(path, content) {
  try {
    await files.writeFile(path, content);
    await files.refresh();
```

```
  } catch (err) {
    console.error('Write failed:', err.message);
    // Fallback to localStorage or download
    localStorage.setItem(`backup_${path}`, content);
  }
}

// Check quota before large operations
if (navigator.storage && navigator.storage.estimate) {
  const estimate = await navigator.storage.estimate();
  const percentUsed = (estimate.usage / estimate.quota) * 100;

  if (percentUsed > 90) {
    console.warn('Storage nearly full, cleanup recommended');
  }
}
```

Exceptions thrown:

- `DOMException` : OPFS-specific errors (NotFoundError, QuotaExceededError)
- `TypeError` : Invalid parameters (non-string path, invalid content type)
- `Error` : General file operation failures

Browser Compatibility Notes:

- OPFS requires Chrome 102+, Edge 102+, Opera 88+
- Not available in Firefox as of December 2024
- Requires HTTPS or localhost (not available on `http://`)
- Private/incognito mode may have reduced quota
- Check support: `if ('storage' in navigator && 'getDirectory' in navigator.storage)`

19.1.7 Common Issues

Files don't persist: OPFS storage clears if user clears browser data, uses private mode, or exceeds quota. Provide export functionality for critical data.

"Failed to initialize OPFS": Requires HTTPS (or localhost). Check browser support: Chrome 102+, Edge 102+, Opera 88+.

File list doesn't update: Call `refresh()` after `writeFile()` or `deleteFile()`.

Can't access files from network: OPFS is origin-private by design. Read content and send via fetch/WebSocket to share.

19.2 pan-markdown-editor

Purpose: Rich markdown editor with toolbar, preview, and auto-save
Import: `<script type="module" src="/ui/pan-markdown-editor.mjs"></script>`

19.2.1 Quick Example

```
<pan-markdown-editor
  value="# Hello World"
  preview="true"
  autosave="true"
  placeholder="Start writing...">
</pan-markdown-editor>

<script type="module">
const editor = document.querySelector('pan-markdown-editor');

// Get content
const markdown = editor.getValue();

// Set content
editor.setValue('# New Content');

// Insert at cursor
editor.insertText('\n\n---\n\n');
</script>
```

19.2.2 Attributes

Attribute	Type	Default	Description
`value`	String	""	Initial markdown content
`placeholder`	String	"Start writing..."	Placeholder text
`preview`	Boolean	false	Show live preview pane
`autosave`	Boolean	false	Enable auto-save (1s debounce)

19.2.3 Methods

Method	Parameters	Returns	Description
`setValue(value)`	value: String	void	Set editor content
`getValue()`	-	String	Get current content
`insertText(text)`	text: String	void	Insert at cursor position
`focus()`	-	void	Focus editor textarea

19.2.4 Toolbar Actions

Button	Action	Shortcut	Result
B	Bold	Ctrl+B	`**text**`
I	Italic	Ctrl+I	`*text*`
S	Strikethrough	-	`~~text~~`
H1-H3	Headings	-	`# text`
*	Bullet list	-	`* item`
1.	Numbered list	-	`1. item`
v	Task list	-	`- [ ] task`
[link]	Link	Ctrl+K	`[text](url)`
[img]	Image	-	`![alt](url)`
{ }	Inline code	-	`` `code` ``
</>	Code block	-	```` ```lang\ncode\n``` ````
”	Blockquote	-	`> quote`
-	Horizontal rule	-	`---`
[+]	Table	-	Markdown table
[eye]	Preview	-	Toggle preview pane

Additional shortcuts: - Ctrl+S: Save (triggers `markdown.saved`) - Tab: Insert two

spaces - Enter: Auto-continue lists

19.2.5 PAN Events

19.2.5.1 Published

Topic	Data	Description
`markdown.changed`	`{ content, wordCount, charCount }`	Content changed
`markdown.saved`	`{ content }`	Ctrl+S pressed or auto-save

19.2.5.2 Subscribed

Topic	Data	Description
`markdown.set-content`	`{ content }`	Set editor content
`markdown.get-content`	`{}`	Request content (responds with markdown.content-response)

19.2.6 Complete Example

```
<!DOCTYPE html>
<html>
<head>
  <script type="module">
    import './pan-bus.mjs';
    import './pan-files.mjs';
    import './pan-markdown-editor.mjs';
    import './pan-markdown-renderer.mjs';

    customElements.whenDefined('pan-bus').then(() => {
      const bus = document.querySelector('pan-bus');
      let currentFile = null;

      // Load file into editor
      bus.subscribe('file.selected', (msg) => {
        if (msg.data.path.endsWith('.md')) {
          currentFile = msg.data.path;
          bus.publish('file.load', { path: msg.data.path });
        }
      });

      bus.subscribe('file.content-loaded', (msg) => {
        const editor = document.querySelector('pan-markdown-
        editor');
        editor.setValue(msg.data.content);
        document.getElementById('filename').textContent =
          msg.data.path.split('/').pop();
      });

      // Save editor to file
      bus.subscribe('markdown.saved', (msg) => {
        if (currentFile) {
```

```
          bus.publish('file.save', {
            path: currentFile,
            content: msg.data.content
          });
        }
      });

      window.createNote = () => {
        currentFile = `/note-${Date.now()}.md`;
        document.getElementById('filename').textContent =
          currentFile.split('/').pop();
        const editor = document.querySelector('pan-markdown-
        editor');
        editor.setValue('# New Note\n\n');
        editor.focus();
      };
    });
  </script>
  <style>
    body { margin: 0; display: grid; grid-template-rows: auto 1fr;
        height: 100vh; }
    .toolbar { padding: 1rem; border-bottom: 1px solid #ccc;
        display: flex; gap: 1rem; }
    .content { display: grid; grid-template-columns: 250px 1fr;
        gap: 1rem; padding: 1rem; }
  </style>
</head>
<body>
  <pan-bus></pan-bus>
  <div class="toolbar">
    <span id="filename">No file selected</span>
    <button onclick="createNote()">New Note</button>
  </div>
  <div class="content">
```

```
    <pan-files filter=".md"></pan-files>
    <pan-markdown-editor preview="true" autosave="true"></pan-
        markdown-editor>
  </div>
</body>
</html>
```

19.2.7 Errors

pan-markdown-editor dispatches `error` events for error conditions:

Error Code	Cause	Resolution
`RENDER_FAILED`	Preview rendering failed	Check markdown syntax or disable preview
`AUTOSAVE_FAILED`	Auto-save operation failed	Check storage or disable autosave
`INVALID_CONTENT`	Non-string content passed to `setValue()`	Pass valid string to `setValue()`
`TOOLBAR_ACTION_FAILED`	Toolbar button action threw error	Check custom toolbar handlers

Error handling example:

```
const editor = document.querySelector('pan-markdown-editor');

// Listen for errors
editor.addEventListener('error', (e) => {
  const { code, message, action } = e.detail;
  console.error(`Editor error [${code}]:`, message);

  switch (code) {
    case 'RENDER_FAILED':
      // Disable preview temporarily
      editor.removeAttribute('preview');
      showNotification('Preview disabled due to rendering error');
      break;

    case 'AUTOSAVE_FAILED':
      // Manual save prompt
      if (confirm('Auto-save failed. Save manually?')) {
        const content = editor.getValue();
        manualSave(content);
      }
      break;

    case 'INVALID_CONTENT':
      console.warn('Attempted to set invalid content');
      break;
  }
});

// Safe setValue with validation
function safeSetValue(content) {
  if (typeof content !== 'string') {
    console.error('Content must be a string');
```

```
    return;
  }

  try {
    editor.setValue(content);
  } catch (err) {
    console.error('Failed to set editor content:', err);
    // Fallback to plain textarea
    useFallbackEditor(content);
  }
}

// Monitor autosave with error handling
const bus = document.querySelector('pan-bus');
bus.subscribe('markdown.saved', (msg) => {
  try {
    localStorage.setItem('backup', msg.data.content);
    console.log('Content backed up');
  } catch (err) {
    console.error('Backup failed:', err);
  }
});
```

Exceptions thrown:

- `TypeError`: Invalid parameters (non-string content, invalid position)
- `Error`: General operation failures (toolbar actions, preview rendering)

Performance Notes:

- Documents over 50KB may cause lag with preview enabled
- Autosave debounces input by 1 second
- Consider disabling preview for large documents: `if (content.length > 50000) editor.removeAttribute('preview')`

19.2.8 Common Issues

Toolbar doesn't work on mobile: Component optimized for desktop. Hide toolbar on mobile.

Preview doesn't update: Ensure `pan-markdown-renderer` is imported.

Large documents lag: Disable preview or add debouncing for documents over 50KB.

Keyboard shortcuts conflict: Component uses `preventDefault()` but browser behavior varies.

19.3 pan-markdown-renderer

Purpose: Render markdown as formatted HTML with syntax highlighting structure
Import: `<script type="module" src="/ui/pan-markdown-renderer.mjs"></script>`

19.3.1 Quick Example

```
<pan-markdown-renderer content="# Hello World

This is **bold** and *italic*.

```javascript
console.log('Code block');
```

">
```

```
### Attributes

| Attribute | Type | Default | Description |
|-----------|------|---------|-------------|
| `content` | String | "" | Markdown content to render |
| `sanitize` | Boolean | true | Escape raw HTML (disable only if
trusted) |

### Methods

| Method | Parameters | Returns | Description |
|--------|-----------|---------|-------------|
| `setContent(content)` | content: String | void | Set and render
markdown |
| `getContent()` | - | String | Get current markdown |
| `getHtml()` | - | String | Get rendered HTML output |

### Supported Syntax

**Standard markdown:**
- Headings: `# H1` through `###### H6`
- Emphasis: `**bold**`, `*italic*`, `~~strikethrough~~`
- Lists: Bullet (`*`), numbered (`1.`), task (`- [ ]`)
- Links: `[text](url)` and images: `![alt](url)`
- Code: Inline `` `code` `` and fenced ` ```lang ` blocks
- Blockquotes: `> quote`
- Horizontal rules: `---`

**GitHub-flavored:**
- Tables with `|` delimiters
- Task lists: `- [x] Done` and `- [ ] Todo`
```

````
### Styling

Customize with CSS variables:

```css
pan-markdown-renderer {
  --color-text: #1e293b;
  --color-text-muted: #64748b;
  --color-bg-alt: #f8fafc;
  --color-border: #e2e8f0;
  --color-code-bg: #1e293b;
  --color-code-text: #e2e8f0;
  --color-primary: #006699;
  --font-mono: 'Courier New', monospace;
}

/* Dark mode */
pan-markdown-renderer.dark {
  --color-text: #e2e8f0;
  --color-bg-alt: #1e293b;
  --color-code-bg: #0f172a;
  --color-primary: #38bdf8;
}
````
````

19.3.2 PAN Events

19.3.2.1 Subscribed

Topic	Data	Description
`markdown.render`	`{ content }`	Render new content

19.3.3 Complete Example

```
<!DOCTYPE html>
<html>
<head>
  <script type="module">
    import './pan-bus.mjs';
    import './pan-markdown-renderer.mjs';

    const docs = {
      intro: `# Getting Started

Welcome! This guide covers the basics.

## Prerequisites

- Modern web browser
- Basic HTML/JavaScript knowledge
- 15 minutes

## Installation

\`\`\`javascript
import './components/app.mjs';
\`\`\`,

      api: `# API Reference

## Core Methods

### \`initialize(config)\`

Initializes the application.
```

```
**Parameters:**
- \`config.debug\` (Boolean): Enable debug mode
- \`config.theme\` (String): Theme name

\`\`\`javascript
await initialize({ debug: true, theme: 'dark' });
\`\`\`,

      examples: `# Examples

## Task List

- [x] Create component
- [x] Write docs
- [ ] Deploy

## Data Table

| Feature | Status | Version |
|---------|--------|---------|
| Import  | ✓      | 1.0     |
| Export  | ✓      | 1.0     |
| Sync    | ⏳     | 2.0     |`
    };

    customElements.whenDefined('pan-bus').then(() => {
      const renderer = document.querySelector('pan-markdown-
        renderer');

      window.showDoc = (section) => {
        renderer.setContent(docs[section]);
        document.querySelectorAll('nav button').forEach(btn => {
```

```
          btn.classList.toggle('active', btn.dataset.section ===
        section);
        });
      };

      showDoc('intro');
    });
  </script>
  <style>
    body { margin: 0; display: grid; grid-template-columns: 200px
        1fr; height: 100vh; }
    nav { background: #f8fafc; padding: 1rem; border-right: 1px
        solid #e2e8f0; }
    nav button { display: block; width: 100%; padding: 0.5rem;
        margin: 0.25rem 0;
                 border: none; background: transparent; text-align:
        left; cursor: pointer; }
    nav button:hover { background: #e2e8f0; }
    nav button.active { background: #006699; color: white; }
    main { padding: 2rem; overflow-y: auto; }
  </style>
</head>
<body>
  <pan-bus></pan-bus>
  <nav>
    <h2>Documentation</h2>
    <button data-section="intro" onclick="showDoc('intro')"
        class="active">Getting Started</button>
    <button data-section="api" onclick="showDoc('api')">API
        Reference</button>
    <button data-section="examples"
        onclick="showDoc('examples')">Examples</button>
  </nav>
  <main>
    <pan-markdown-renderer></pan-markdown-renderer>
  </main>
```

```
</body>
</html>
```

19.3.4 Errors

pan-markdown-renderer dispatches `error` events when rendering fails:

Error Code	Cause	Resolution
`PARSE_ERROR`	Invalid markdown syntax	Check markdown format or sanitize input
`RENDER_ERROR`	HTML generation failed	Simplify content or report bug
`SANITIZATION_ERROR`	HTML sanitization failed	Check content or disable sanitization
`INVALID_CONTENT`	Non-string content passed	Pass valid string to `setContent()`

Error handling example:

```
const renderer = document.querySelector('pan-markdown-renderer');

// Listen for errors
renderer.addEventListener('error', (e) => {
  const { code, message, content } = e.detail;
  console.error(`Renderer error [${code}]:`, message);

  switch (code) {
    case 'PARSE_ERROR':
      // Show raw markdown as fallback
      renderer.shadowRoot.querySelector('.content').textContent =
        content;
      showWarning('Markdown parsing failed, showing raw content');
      break;

    case 'RENDER_ERROR':
      console.error('Failed to render markdown');
      showFallbackRenderer();
      break;

    case 'SANITIZATION_ERROR':
      // Retry with stricter sanitization
      renderer.setAttribute('sanitize', 'true');
      break;

    case 'INVALID_CONTENT':
      console.warn('Invalid content type provided');
      break;
  }
});

// Safe setContent with validation
```

```
function safeSetContent(content) {
  if (typeof content !== 'string') {
    console.error('Content must be a string');
    return;
  }

  try {
    renderer.setContent(content);
  } catch (err) {
    console.error('Failed to render content:', err);
    // Fallback to plain text
    renderer.shadowRoot.querySelector('.content').textContent =
        content;
  }
}

// Validate markdown before rendering
function validateAndRender(markdown) {
  // Check for balanced code fences
  const fenceCount = (markdown.match(/```/g) || []).length;
  if (fenceCount % 2 !== 0) {
    console.warn('Unbalanced code fences detected');
    markdown += '\n```'; // Auto-close
  }

  renderer.setContent(markdown);
}
```

Exceptions thrown:

- `TypeError` : Invalid parameters (non-string content)
- `Error` : Parse or render failures

Performance Notes:

- Large documents (>100KB) may slow rendering
- Complex tables with many rows cause layout reflow
- Consider debouncing content updates for live preview

19.3.5 Common Issues

No syntax highlighting: Renderer provides structure only. Add Prism.js or similar for color syntax highlighting.

Tables render incorrectly: Ensure separator row with dashes: `|---|---|`

Raw HTML appears: Sanitization enabled by default. Only set `sanitize="false"` for trusted content.

Content doesn't wrap on mobile: Add responsive CSS:

```
pan-markdown-renderer { overflow-x: auto; }
pan-markdown-renderer pre { max-width: 100%; overflow-x: auto; }
```

19.4 Summary

This chapter documented LARC's UI components:

- **pan-files**: OPFS file system browser with visual UI and programmatic API
- **pan-markdown-editor**: Rich markdown editor with toolbar, preview, and auto-save
- **pan-markdown-renderer**: Markdown-to-HTML renderer with GitHub-flavored syntax

Use together for complete markdown applications or individually as needed.

See Also:

- Tutorial: *Learning LARC* Chapter 10
- Core components: Chapter 17
- Data components: Chapter 18
- Integration patterns: Chapter 20

20 Integration Components Reference

API documentation for LARC's integration components. For tutorials, see *Learning LARC* Chapter 11.

20.1 pan-data-connector

Purpose: Declarative REST API bridge with CRUD pattern
Import: `<script type="module" src="/ui/pan-data-connector.mjs"></script>`

20.1.1 Quick Example

```
<pan-data-connector
  resource="users"
  base-url="https://api.example.com"
  key="id">
</pan-data-connector>

<script type="module">
import { PanClient } from './pan-client.mjs';

const pc = new PanClient();

// Fetch list
pc.publish('users.list.get', { page: 1, limit: 20 });

// Listen for results
pc.subscribe('users.list.state', (msg) => {
  console.log('Users:', msg.data.items);
});

// Get single item
pc.publish('users.item.get', { id: 123 });

// Save item
pc.publish('users.item.save', {
  id: 123,
  name: 'Alice',
  email: 'alice@example.com'
});
</script>
```

20.1.2 Attributes

Attribute	Type	Default	Description
`resource`	String	“items”	Resource name (topic prefix)
`base-url`	String	“”	API base URL
`key`	String	“id”	Unique identifier field
`list-path`	String	“/${resource}”	List endpoint template
`item-path`	String	“/${resource}/:id”	Item endpoint template
`update-method`	String	“PUT”	HTTP method for updates (PUT/PATCH)
`credentials`	String	“”	Fetch credentials mode

Headers configuration (via `<script type="application/json">`):

```
<pan-data-connector resource="users" base-url="/api">
  <script type="application/json">
    {
      "headers": {
        "Authorization": "Bearer TOKEN",
        "X-API-Version": "2023-01"
      }
    }
  </script>
</pan-data-connector>
```

20.1.3 PAN Topics

20.1.3.1 Subscribed (Requests)

Topic	Data	HTTP Request	Description
`\${resource}.list.get`	`{ ...query }`	GET /${resource}	Fetch list
`\${resource}.item.get`	`{ id }`	GET /${resource}/:id	Fetch single item
`\${resource}.item.save`	`{ id?, ...data }`	POST or PUT	Create/update item
`\${resource}.item.delete`	`{ id }`	DELETE /${resource}/:id	Delete item

20.1.3.2 Published (Responses)

Topic	Data	Description
`\${resource}.list.state`	`{ items: [...], meta? }`	List results (retained)
`\${resource}.item.state.\${id}`	`{ ...item }`	Single item (retained)
`\${resource}.error`	`{ operation, error, status }`	Error occurred

20.1.4 Complete Example

```
<!DOCTYPE html>
<html>
<head>
  <script type="module">
    import './pan-bus.mjs';
    import './pan-data-connector.mjs';
    import { PanClient } from './pan-client.mjs';

    customElements.whenDefined('pan-bus').then(() => {
      const pc = new PanClient();

      // Render user list
      pc.subscribe('users.list.state', (msg) => {
        const list = document.getElementById('user-list');
        list.innerHTML = msg.data.items.map(user => `
          <li>
            ${user.name} (${user.email})
            <button onclick="editUser(${user.id})">Edit</button>
            <button onclick="deleteUser(${user.id})">Delete</button>
          </li>
        `).join('');
      });

      // Handle errors
      pc.subscribe('users.error', (msg) => {
        console.error('API Error:', msg.data.error);
        alert(`Error: ${msg.data.error}`);
      });

      // Load users on startup
      pc.publish('users.list.get', {});
```

```
      // CRUD operations
      window.editUser = (id) => {
        pc.publish('users.item.get', { id });
        pc.subscribe('users.item.state.' + id, (msg) => {
          document.getElementById('name').value = msg.data.name;
          document.getElementById('email').value = msg.data.email;
          document.getElementById('user-id').value = msg.data.id;
        });
      };

      window.saveUser = () => {
        const id = document.getElementById('user-id').value;
        pc.publish('users.item.save', {
          id: id || undefined,
          name: document.getElementById('name').value,
          email: document.getElementById('email').value
        });
      };

      window.deleteUser = (id) => {
        if (confirm('Delete user?')) {
          pc.publish('users.item.delete', { id });
          setTimeout(() => pc.publish('users.list.get', {}), 100);
        }
      };
    });
  </script>
</head>
<body>
  <pan-bus></pan-bus>
  <pan-data-connector resource="users" base-url="/api"
        key="id"></pan-data-connector>
```

```
  <div>
    <h2>Users</h2>
    <ul id="user-list"></ul>

    <h3>Edit User</h3>
    <input type="hidden" id="user-id">
    <input id="name" placeholder="Name">
    <input id="email" placeholder="Email">
    <button onclick="saveUser()">Save</button>
  </div>
</body>
</html>
```

20.1.5 Errors

pan-data-connector publishes errors to `${resource}.error` topic:

Error Condition	Cause	Resolution
`NETWORK_ERROR`	Fetch failed (offline, DNS, timeout)	Check network connection and API availability
`HTTP_4XX`	Client error (400-499)	Verify request data and authentication
`HTTP_5XX`	Server error (500-599)	Check API server logs, retry with backoff
`PARSE_ERROR`	Response not valid JSON	Check API response format
`CORS_ERROR`	Cross-origin request blocked	Configure CORS headers on server
`INVALID_CONFIG`	Missing required attributes	Set `resource` and `base-url` attributes

Error handling example:

```
import { PanClient } from './pan-client.mjs';

const pc = new PanClient();

// Subscribe to errors
pc.subscribe('users.error', (msg) => {
  const { operation, error, status, response } = msg.data;
  console.error(`API Error [${operation}]:`, error);

  switch (true) {
    case status === 401:
      // Unauthorized - redirect to login
      window.location.href = '/login';
      break;

    case status === 404:
      // Not found
      showNotification('Resource not found');
      break;

    case status >= 500:
      // Server error - retry with exponential backoff
      retryWithBackoff(() => {
        pc.publish('users.list.get', {});
      });
      break;

    case error.includes('CORS'):
      // CORS error
      console.error('CORS configuration required on server');
      showCORSHelp();
      break;
```

```
    case error.includes('NETWORK'):
      // Network error
      showOfflineMode();
      break;

    default:
      showNotification(`Error: ${error}`);
  }
});

// Retry with exponential backoff
let retryCount = 0;
function retryWithBackoff(fn, maxRetries = 3) {
  if (retryCount >= maxRetries) {
    console.error('Max retries exceeded');
    return;
  }
  const delay = Math.pow(2, retryCount) * 1000;
  setTimeout(() => {
    retryCount++;
    fn();
  }, delay);
}

// Optimistic updates with rollback on error
let previousState = null;
pc.subscribe('users.list.state', (msg) => {
  previousState = msg.data;
});

function saveUserOptimistically(user) {
  // Update UI immediately
```

```
  updateUI(user);

  // Save to API
  pc.publish('users.item.save', user);

  // Rollback on error
  const unsubscribe = pc.subscribe('users.error', (msg) => {
    if (msg.data.operation === 'save') {
      updateUI(previousState);
      unsubscribe();
    }
  });
}
```

HTTP Status Codes: All HTTP status codes are passed through in the error message:

- 400: Bad Request
- 401: Unauthorized
- 403: Forbidden
- 404: Not Found
- 422: Unprocessable Entity
- 500: Internal Server Error
- 502: Bad Gateway
- 503: Service Unavailable

20.1.6 Common Issues

401/403 errors: Configure authentication headers via embedded JSON script.

CORS errors: Ensure API server sets appropriate CORS headers. Use `credentials="include"` for cookies.

List doesn't update after save: Connector doesn't auto-refresh lists. Manually

request: `pc.publish('${resource}.list.get', {})`.

Non-standard API: Override `list-path` and `item-path` attributes for custom endpoints.

20.2 pan-graphql-connector

Purpose: GraphQL query and mutation bridge to PAN bus
Import: `<script type="module" src="/ui/pan-graphql-connector.mjs"></script>`

20.2.1 Quick Example

```
<pan-graphql-connector
  endpoint="https://api.example.com/graphql"
  resource="users">
  <script type="application/graphql" data-operation="list">
    query GetUsers($limit: Int) {
      users(limit: $limit) {
        id
        name
        email
      }
    }
  </script>

  <script type="application/graphql" data-operation="get">
    query GetUser($id: ID!) {
      user(id: $id) {
        id
        name
        email
        createdAt
      }
    }
  </script>
</pan-graphql-connector>

<script type="module">
import { PanClient } from './pan-client.mjs';

const pc = new PanClient();

// Execute query
pc.publish('users.list.get', { limit: 10 });
```

```
pc.subscribe('users.list.state', (msg) => {
  console.log('Users:', msg.data);
});
</script>
```

20.2.2 Attributes

Attribute	Type	Default	Description
`endpoint`	String	“/graphql”	GraphQL endpoint URL
`resource`	String	“items”	Resource name (topic prefix)
`key`	String	“id”	Unique identifier field

GraphQL operations (via `<script type="application/graphql">`):

- `data-operation="list"`: List query
- `data-operation="get"`: Single item query
- `data-operation="create"`: Create mutation
- `data-operation="update"`: Update mutation
- `data-operation="delete"`: Delete mutation

Headers configuration (via `<script type="application/json">`):

```
<pan-graphql-connector endpoint="/graphql" resource="users">
  <script type="application/json">
    {
      "headers": {
        "Authorization": "Bearer TOKEN"
      }
    }
  </script>
  <!-- GraphQL scripts... -->
</pan-graphql-connector>
```

20.2.3 PAN Topics

Same as `pan-data-connector`:

- Subscribe: `${resource}.list.get`, `${resource}.item.get`, `${resource}.item.save`, `${resource}.item.delete`
- Publish: `${resource}.list.state`, `${resource}.item.state.${id}`, `${resource}.error`

20.2.4 Complete Example

```
<!DOCTYPE html>
<html>
<head>
  <script type="module">
    import './pan-bus.mjs';
    import './pan-graphql-connector.mjs';
    import { PanClient } from './pan-client.mjs';

    customElements.whenDefined('pan-bus').then(() => {
      const pc = new PanClient();

      // Fetch users
      pc.publish('users.list.get', { limit: 20 });

      // Display results
      pc.subscribe('users.list.state', (msg) => {
        document.getElementById('output').textContent =
          JSON.stringify(msg.data, null, 2);
      });

      // Handle errors
      pc.subscribe('users.error', (msg) => {
        console.error('GraphQL Error:', msg.data.error);
      });
    });
  </script>
</head>
<body>
  <pan-bus></pan-bus>

  <pan-graphql-connector endpoint="https://api.example.com/graphql"
        resource="users">
```

```
<script type="application/json">
  { "headers": { "Authorization": "Bearer TOKEN" } }
</script>

<script type="application/graphql" data-operation="list">
  query GetUsers($limit: Int) {
    users(limit: $limit) {
      id
      name
      email
    }
  }
</script>

<script type="application/graphql" data-operation="get">
  query GetUser($id: ID!) {
    user(id: $id) {
      id
      name
      email
      createdAt
    }
  }
</script>

<script type="application/graphql" data-operation="create">
  mutation CreateUser($input: UserInput!) {
    createUser(input: $input) {
      id
      name
      email
    }
  }
```

```
    </script>
  </pan-graphql-connector>

  <button onclick="pc.publish('users.list.get', { limit: 10 })">Load
        Users</button>
  <pre id="output"></pre>
</body>
</html>
```

20.2.5 Errors

pan-graphql-connector publishes errors to `${resource}.error` topic:

Error Condition	Cause	Resolution
`GRAPHQL_ERROR`	GraphQL errors in response	Check query syntax and schema
`NETWORK_ERROR`	Fetch failed	Check network and endpoint URL
`PARSE_ERROR`	Response not valid JSON	Check GraphQL endpoint response format
`MISSING_OPERATION`	No query/mutation defined for operation	Add `<script type="application/graphql" data-operation="...">`
`VALIDATION_ERROR`	GraphQL validation failed	Fix query variables or schema mismatch
`AUTH_ERROR`	Authentication failed	Check Authorization header

Error handling example:

```
import { PanClient } from './pan-client.mjs';

const pc = new PanClient();

// Subscribe to GraphQL errors
pc.subscribe('users.error', (msg) => {
  const { operation, error, graphqlErrors, status } = msg.data;
  console.error(`GraphQL Error [${operation}]:`, error);

  // Handle GraphQL-specific errors
  if (graphqlErrors && graphqlErrors.length > 0) {
    graphqlErrors.forEach(err => {
      console.error(`  - ${err.message}`);

      // Handle specific error types
      if (err.extensions?.code === 'UNAUTHENTICATED') {
        redirectToLogin();
      } else if (err.extensions?.code === 'FORBIDDEN') {
        showNotification('Access denied');
      } else if (err.path) {
        showFieldError(err.path.join('.'), err.message);
      }
    });
  }

  // Handle network errors
  if (error.includes('NETWORK')) {
    showOfflineMode();
  }

  // Handle missing operations
  if (error.includes('MISSING_OPERATION')) {
```

```
    console.error(`Add <script type="application/graphql" data-
        operation="${operation}"> to component`);
  }
});

// Retry with different variables on error
function retryQueryWithFallback() {
  const unsubscribe = pc.subscribe('users.error', (msg) => {
    if (msg.data.operation === 'list') {
      // Try with reduced limit
      pc.publish('users.list.get', { limit: 10 });
      unsubscribe();
    }
  });

  pc.publish('users.list.get', { limit: 100 });
}

// Handle partial data from GraphQL
pc.subscribe('users.list.state', (msg) => {
  const { items, errors } = msg.data;

  if (errors) {
    // GraphQL can return partial data with errors
    console.warn('Partial data received with errors:', errors);
    showWarning('Some data may be incomplete');
  }

  displayUsers(items || []);
});
```

GraphQL Error Extensions: GraphQL servers often include error codes in `extensions`:

- `UNAUTHENTICATED`: Not logged in
- `FORBIDDEN`: Insufficient permissions
- `BAD_USER_INPUT`: Invalid variables
- `GRAPHQL_VALIDATION_FAILED`: Query validation error
- `PERSISTED_QUERY_NOT_FOUND`: Persisted query ID not found

20.2.6 Common Issues

Query not found: Ensure `data-operation` attribute matches operation type (list/get/create/update/delete).

Variables not passed: Variable names must match GraphQL schema. Check operation definitions.

Response path mapping: Use `data-path` attribute to specify nested result path: `data-path="data.users"`.

20.3 pan-websocket

Purpose: Bidirectional WebSocket communication via PAN bus
Import: `<script type="module" src="/ui/pan-websocket.mjs"></script>`

20.3.1 Quick Example

```
<pan-websocket
  url="wss://api.example.com/ws"
  auto-connect="true">
</pan-websocket>

<script type="module">
import { PanClient } from './pan-client.mjs';

const pc = new PanClient();

// Listen for connection
pc.subscribe('ws.connected', (msg) => {
  console.log('WebSocket connected');
});

// Send message
pc.publish('ws.send', {
  type: 'subscribe',
  channel: 'notifications'
});

// Receive messages
pc.subscribe('ws.message', (msg) => {
  console.log('Received:', msg.data);
});
</script>
```

20.3.2 Attributes

Attribute	Type	Default	Description
`url`	String	“”	WebSocket URL (ws:// or wss://)
`auto-connect`	Boolean	false	Connect on component mount
`auto-reconnect`	Boolean	true	Reconnect on disconnect
`reconnect-delay`	Number	1000	Reconnect delay (ms)
`max-reconnect-attempts`	Number	Infinity	Max reconnection attempts

20.3.3 Methods

Method	Parameters	Returns	Description
`connect()`	-	Promise<void>	Open WebSocket connection
`disconnect()`	-	void	Close connection
`send(data)`	data: Any	void	Send message (auto-serializes JSON)

20.3.4 PAN Topics

20.3.4.1 Published

Topic	Data	Description
`ws.connected`	`{ url }`	Connection opened
`ws.disconnected`	`{ code, reason }`	Connection closed
`ws.message`	`{ ...data }`	Message received
`ws.error`	`{ error }`	Error occurred

20.3.4.2 Subscribed

Topic	Data	Description
`ws.connect`	`{ url? }`	Open connection
`ws.disconnect`	`{}`	Close connection
`ws.send`	`{ ...data }`	Send message

20.3.5 Complete Example

```
<!DOCTYPE html>
<html>
<head>
  <script type="module">
    import './pan-bus.mjs';
    import './pan-websocket.mjs';
    import { PanClient } from './pan-client.mjs';

    customElements.whenDefined('pan-bus').then(() => {
      const pc = new PanClient();

      // Connection status
      pc.subscribe('ws.connected', (msg) => {
        document.getElementById('status').textContent = 'Connected';
        document.getElementById('status').style.color = 'green';
      });

      pc.subscribe('ws.disconnected', (msg) => {
        document.getElementById('status').textContent =
        'Disconnected';
        document.getElementById('status').style.color = 'red';
      });

      // Receive messages
      pc.subscribe('ws.message', (msg) => {
        const log = document.getElementById('log');
        const entry = document.createElement('div');
        entry.textContent = JSON.stringify(msg.data);
        log.prepend(entry);
      });

      // Send message
```

```
      window.sendMessage = () => {
        const input = document.getElementById('message');
        pc.publish('ws.send', {
          type: 'chat',
          message: input.value,
          timestamp: Date.now()
        });
        input.value = '';
      };

      // Connect/disconnect
      window.connect = () => pc.publish('ws.connect', {});
      window.disconnect = () => pc.publish('ws.disconnect', {});
    });
  </script>
</head>
<body>
  <pan-bus></pan-bus>
  <pan-websocket url="wss://api.example.com/ws" auto-
        connect="false"></pan-websocket>

  <div>
    <h2>WebSocket Demo</h2>
    Status: <span id="status">Disconnected</span>
    <button onclick="connect()">Connect</button>
    <button onclick="disconnect()">Disconnect</button>

    <div>
      <input id="message" placeholder="Enter message">
      <button onclick="sendMessage()">Send</button>
    </div>

    <h3>Log</h3>
```

```
    <div id="log"></div>
  </div>
</body>
</html>
```

20.3.6 Errors

pan-websocket publishes errors to `ws.error` topic:

Error Condition	Cause	Resolution
`CONNECTION_FAILED`	Cannot connect to WebSocket server	Check URL and server availability
`CONNECTION_CLOSED`	Connection closed unexpectedly	Check server logs, will auto-reconnect if enabled
`SEND_FAILED`	Message send failed	Check connection status before sending
`INVALID_URL`	WebSocket URL is invalid	Use valid ws:// or wss:// URL
`MAX_RECONNECT_EXCEEDED`	Max reconnection attempts reached	Check server, increase max attempts, or fix issue
`MESSAGE_PARSE_ERROR`	Cannot parse received message	Check message format from server

Error handling example:

```
import { PanClient } from './pan-client.mjs';

const pc = new PanClient();

// Subscribe to errors
pc.subscribe('ws.error', (msg) => {
  const { error, code, url } = msg.data;
  console.error('WebSocket Error:', error);

  switch (code) {
    case 1006: // Abnormal closure
      console.warn('Connection lost abnormally');
      showNotification('Connection lost, reconnecting...');
      break;

    case 1008: // Policy violation
      console.error('Connection rejected by server');
      showNotification('Access denied');
      break;

    case 1011: // Server error
      console.error('Server error occurred');
      retryAfterDelay(5000);
      break;

    default:
      if (error.includes('MAX_RECONNECT_EXCEEDED')) {
        showNotification('Cannot connect to server');
        showOfflineMode();
      }
  }
});
```

```
// Monitor connection lifecycle
let connectionAttempts = 0;

pc.subscribe('ws.connected', (msg) => {
  connectionAttempts = 0;
  console.log('WebSocket connected');
  showOnlineStatus();
});

pc.subscribe('ws.disconnected', (msg) => {
  const { code, reason } = msg.data;
  console.log(`Disconnected: ${code} - ${reason}`);

  connectionAttempts++;
  if (connectionAttempts > 3) {
    showPersistentConnectionIssueWarning();
  }
});

// Heartbeat to detect connection issues
let heartbeatInterval;
let lastHeartbeat = Date.now();

pc.subscribe('ws.connected', () => {
  heartbeatInterval = setInterval(() => {
    pc.publish('ws.send', { type: 'ping', timestamp: Date.now() });

    // Check if we've received pong
    if (Date.now() - lastHeartbeat > 30000) {
      console.warn('Heartbeat timeout, reconnecting...');
      pc.publish('ws.disconnect', {});
      setTimeout(() => pc.publish('ws.connect', {}), 1000);
```

```
    }
  }, 10000);
});

pc.subscribe('ws.message', (msg) => {
  if (msg.data.type === 'pong') {
    lastHeartbeat = Date.now();
  }
});

pc.subscribe('ws.disconnected', () => {
  clearInterval(heartbeatInterval);
});

// Message queuing when offline
const messageQueue = [];

function sendMessage(data) {
  // Check if connected
  if (document.querySelector('pan-websocket').readyState ===
      WebSocket.OPEN) {
    pc.publish('ws.send', data);
  } else {
    // Queue message
    messageQueue.push(data);
    console.log('Message queued (offline)');
  }
}

// Send queued messages on reconnect
pc.subscribe('ws.connected', () => {
  while (messageQueue.length > 0) {
    const data = messageQueue.shift();
```

```
    pc.publish('ws.send', data);
  }
});
```

WebSocket Close Codes: Standard close codes passed in `ws.disconnected`:

- 1000: Normal closure
- 1001: Going away (page unload)
- 1002: Protocol error
- 1003: Unsupported data
- 1006: Abnormal closure (no close frame)
- 1007: Invalid data
- 1008: Policy violation
- 1009: Message too big
- 1011: Server error

20.3.7 Common Issues

Connection fails: Check URL scheme (ws:// for HTTP, wss:// for HTTPS). Ensure server is running.

Auto-reconnect loops: Increase `reconnect-delay` or set `max-reconnect-attempts` to prevent rapid retries.

Messages not received: Ensure message format matches expected structure. Check `ws.message` subscription.

CORS issues: WebSocket doesn't use CORS. Check server's Origin header validation.

20.4 pan-sse

Purpose: Server-Sent Events streaming via PAN bus
Import: `<script type="module" src="/ui/pan-sse.mjs"></script>`

20.4.1 Quick Example

```
<pan-sse
  url="https://api.example.com/events"
  auto-connect="true">
</pan-sse>

<script type="module">
import { PanClient } from './pan-client.mjs';

const pc = new PanClient();

// Listen for connection
pc.subscribe('sse.connected', (msg) => {
  console.log('SSE connected');
});

// Receive events
pc.subscribe('sse.message', (msg) => {
  console.log('Event:', msg.data);
});

// Receive custom event types
pc.subscribe('sse.event.notification', (msg) => {
  console.log('Notification:', msg.data);
});
</script>
```

20.4.2 Attributes

Attribute	Type	Default	Description
`url`	String	“”	SSE endpoint URL
`auto-connect`	Boolean	false	Connect on mount
`with-credentials`	Boolean	false	Include credentials in request

20.4.3 Methods

Method	Parameters	Returns	Description
`connect()`	-	void	Open SSE connection
`disconnect()`	-	void	Close connection

20.4.4 PAN Topics

20.4.4.1 Published

Topic	Data	Description
`sse.connected`	`{ url }`	Connection opened
`sse.disconnected`	`{}`	Connection closed
`sse.message`	`{ data, id?, type? }`	Default message event
`sse.event.\${type}`	`{ data, id? }`	Custom event type
`sse.error`	`{ error }`	Error occurred

20.4.4.2 Subscribed

Topic	Data	Description
`sse.connect`	`{ url? }`	Open connection
`sse.disconnect`	`{}`	Close connection

20.4.5 Complete Example

```
<!DOCTYPE html>
<html>
<head>
  <script type="module">
    import './pan-bus.mjs';
    import './pan-sse.mjs';
    import { PanClient } from './pan-client.mjs';

    customElements.whenDefined('pan-bus').then(() => {
      const pc = new PanClient();

      // Connection status
      pc.subscribe('sse.connected', (msg) => {
        document.getElementById('status').textContent = 'Streaming';
        document.getElementById('status').style.color = 'green';
      });

      pc.subscribe('sse.disconnected', (msg) => {
        document.getElementById('status').textContent = 'Stopped';
        document.getElementById('status').style.color = 'red';
      });

      // Receive default messages
      pc.subscribe('sse.message', (msg) => {
        addLog('message', msg.data);
      });

      // Receive custom event types
      pc.subscribe('sse.event.update', (msg) => {
        addLog('update', msg.data);
      });
```

```
      pc.subscribe('sse.event.notification', (msg) => {
        addLog('notification', msg.data);
      });

      function addLog(type, data) {
        const log = document.getElementById('log');
        const entry = document.createElement('div');
        entry.innerHTML = `<strong>${type}:</strong>
        ${JSON.stringify(data)}`;
        log.prepend(entry);
      }

      window.startStream = () => pc.publish('sse.connect', {});
      window.stopStream = () => pc.publish('sse.disconnect', {});
    });
  </script>
</head>
<body>
  <pan-bus></pan-bus>
  <pan-sse url="https://api.example.com/events" auto-
        connect="false"></pan-sse>

  <div>
    <h2>SSE Demo</h2>
    Status: <span id="status">Stopped</span>
    <button onclick="startStream()">Start</button>
    <button onclick="stopStream()">Stop</button>

    <h3>Event Log</h3>
    <div id="log"></div>
  </div>
</body>
</html>
```

20.4.6 Errors

pan-sse publishes errors to `sse.error` topic:

Error Condition	Cause	Resolution
`CONNECTION_FAILED`	Cannot connect to SSE endpoint	Check URL and server availability
`CONNECTION_CLOSED`	Stream closed unexpectedly	Check server logs, SSE auto-reconnects
`PARSE_ERROR`	Cannot parse event data	Check server event format
`INVALID_URL`	SSE URL is invalid	Use valid HTTP/HTTPS URL
`NOT_SUPPORTED`	Browser doesn't support SSE	Use modern browser or provide fallback

Error handling example:

```
import { PanClient } from './pan-client.mjs';

const pc = new PanClient();

// Subscribe to errors
pc.subscribe('sse.error', (msg) => {
  const { error, readyState, url } = msg.data;
  console.error('SSE Error:', error);

  if (error.includes('CONNECTION_FAILED')) {
    // Retry with exponential backoff
    setTimeout(() => {
      pc.publish('sse.connect', { url });
    }, 5000);
  }

  if (error.includes('NOT_SUPPORTED')) {
    // Fall back to polling
    console.warn('SSE not supported, using polling fallback');
    startPolling();
  }

  if (error.includes('PARSE_ERROR')) {
    // Log parse errors but continue streaming
    console.error('Invalid event format, skipping');
  }
});

// Monitor connection lifecycle
pc.subscribe('sse.connected', (msg) => {
  console.log('SSE streaming from:', msg.data.url);
  showStreamingStatus(true);
```

```
});

pc.subscribe('sse.disconnected', () => {
  console.log('SSE stream stopped');
  showStreamingStatus(false);
});

// Handle connection drops gracefully
let eventCount = 0;
let lastEventTime = Date.now();

pc.subscribe('sse.message', (msg) => {
  eventCount++;
  lastEventTime = Date.now();
});

// Check for stale connection
setInterval(() => {
  const timeSinceLastEvent = Date.now() - lastEventTime;

  if (timeSinceLastEvent > 60000) { // 60 seconds
    console.warn('No events received recently, reconnecting...');
    pc.publish('sse.disconnect', {});
    setTimeout(() => pc.publish('sse.connect', {}), 1000);
  }
}, 30000);

// Fallback to polling if SSE unavailable
function startPolling() {
  setInterval(async () => {
    try {
      const res = await fetch('/api/events?since=' + lastEventTime);
      const events = await res.json();
```

```
      events.forEach(event => {
        pc.publish('sse.message', event);
      });
    } catch (err) {
      console.error('Polling failed:', err);
    }
  }, 5000);
}

// Graceful degradation for older browsers
if (!('EventSource' in window)) {
  console.warn('SSE not supported, using polling');
  startPolling();
} else {
  pc.publish('sse.connect', { url: '/api/events' });
}
```

Browser Compatibility:

- SSE (EventSource API) supported in all modern browsers
- Not available in IE11 (use polyfill or fallback to polling)
- Check support: `if ('EventSource' in window)`
- Automatic reconnection is built into the EventSource API
- Maximum reconnection delay: typically 3 seconds

Server Requirements:

- Must send `Content-Type: text/event-stream`
- Must send `Cache-Control: no-cache`
- Keep connection open (don't close after each event)
- Send `:\n\n` (comment) every 15-30 seconds to keep alive
- Event format: `data: {...}\n\n` or `event: type\ndata: {...}\n\n`

20.4.7 Common Issues

Connection closes immediately: Server must keep connection open with proper headers: `Content-Type: text/event-stream`, `Cache-Control: no-cache`.

Events not received: Check event format. Default events use `data:` prefix. Custom events use `event: type` followed by `data:`.

CORS errors: Configure CORS headers on server. Use `with-credentials="true"` for authenticated streams.

No reconnection: SSE automatically reconnects. To prevent, call `disconnect()` before page unload.

20.5 Summary

This chapter documented LARC's integration components:

- **pan-data-connector**: REST API integration with CRUD pattern
- **pan-graphql-connector**: GraphQL query and mutation bridge
- **pan-websocket**: Bidirectional WebSocket communication
- **pan-sse**: Server-Sent Events streaming

All components transform external protocols into PAN bus messages, maintaining architectural consistency.

See Also:

- Tutorial: *Learning LARC* Chapter 11
- Core components: Chapter 17
- Data components: Chapter 18
- Authentication patterns: Appendix B

21 Utility Components Reference

API documentation for LARC's utility components. For tutorials, see *Learning LARC* Chapter 14.

21.1 pan-debug

Purpose: Message tracing and debugging for PAN bus introspection
Import: `import { PanDebugManager } from './pan-debug.mjs';`
Global Access: `window.pan.debug` (when using pan-bus)

21.1.1 Quick Example

```
// Access via global
const debug = window.pan.debug;

// Enable tracing
debug.enableTracing({ maxBuffer: 500, sampleRate: 1.0 });

// Query messages
const recentMessages = debug.getTrace({ limit: 10 });
console.log('Recent messages:', recentMessages);

// Find specific messages
const loginMessages = debug.findMessages({
  topic: 'user.login',
  timeRange: { start: Date.now() - 60000 } // Last minute
});

// Export for analysis
const traceData = debug.exportTrace();
console.log('Trace export:', traceData);

// Disable when done
debug.disableTracing();
```

21.1.2 API

21.1.2.1 Constructor

```
new PanDebugManager()
```

Creates debug manager with defaults:

- `enabled`: false
- `maxBuffer`: 1000
- `sampleRate`: 1.0 (100%)

21.1.2.2 Configuration Methods

Method	Parameters	Description
`enableTracing(options)`	options?: `{ maxBuffer?, sampleRate? }`	Enable message tracing
`disableTracing()`	-	Disable tracing and clear buffer
`setSampleRate(rate)`	rate: Number (0.0-1.0)	Set sampling rate
`clearTrace()`	-	Clear trace buffer

options: - `maxBuffer` (Number, default: 1000): Max messages to retain - `sampleRate` (Number, default: 1.0): Sampling rate (0.0-1.0)

21.1.2.3 Query Methods

Method	Parameters	Returns	Description
`getTrace(options)`	options?: `{ limit?, offset? }`	Array	Get trace buffer
`findMessages(query)`	query: Object	Array	Find messages matching criteria
`getStats()`	-	Object	Get statistics
`exportTrace(format)`	format?: String	String/ Object	Export trace data

Query options: - `topic` (String): Match topic pattern - `clientId` (String): Filter by client ID - `timeRange` (Object): `{ start?, end? }` in milliseconds - `limit` (Number): Max results

Stats object:

```
{
  totalMessages: 1523,
  tracedMessages: 1523,
  droppedMessages: 0,
  bufferSize: 1000,
  sampleRate: 1.0,
  enabled: true
}
```

21.1.3 Complete Example

```
import { PanDebugManager } from './pan-debug.mjs';

// Create and enable
const debug = new PanDebugManager();
debug.enableTracing({ maxBuffer: 1000, sampleRate: 0.1 }); // Sample
        10%

// Build debug UI
function buildDebugPanel() {
  const panel = document.createElement('div');
  panel.style.cssText = `
    position: fixed; bottom: 0; right: 0; width: 400px; height:
        300px;
    background: white; border: 1px solid #ccc; overflow: auto;
    font-family: monospace; font-size: 12px; padding: 10px;
  `;

  const refresh = () => {
    const messages = debug.getTrace({ limit: 20 });
    panel.innerHTML = `
      <h3>PAN Bus Debug (${debug.getStats().totalMessages}
        total)</h3>
      <button onclick="window.panDebug.clearTrace()">Clear</button>
      <button
        onclick="window.panDebug.exportToConsole()">Export</button>
      <hr>
      ${messages.map(msg => `
        <div style="margin: 5px 0; border-left: 3px solid #007bff;
        padding-left: 5px;">
          <strong>${msg.topic}</strong>
          <small>(${new
        Date(msg.timestamp).toLocaleTimeString()})</small>
          <pre style="margin: 2px 0;">${JSON.stringify(msg.data,
        null, 2)}</pre>
        </div>
```

```
        `).join('')}
      `;
    };

    // Refresh every 2 seconds
    setInterval(refresh, 2000);
    refresh();

    document.body.appendChild(panel);

    // Expose controls
    window.panDebug = {
      clearTrace: () => { debug.clearTrace(); refresh(); },
      exportToConsole: () => console.log(debug.exportTrace())
    };
}

// Initialize when bus ready
if (window.__panReady) {
    buildDebugPanel();
} else {
    window.addEventListener('pan:sys.ready', buildDebugPanel, { once:
          true });
}
```

21.1.4 Helper Functions

```
// Filter by topic pattern
function findByTopic(pattern) {
  return debug.findMessages({ topic: pattern });
}

// Get message frequency
function getMessageFrequency() {
  const messages = debug.getTrace();
  const freq = {};
  messages.forEach(msg => {
    freq[msg.topic] = (freq[msg.topic] || 0) + 1;
  });
  return Object.entries(freq).sort((a, b) => b[1] - a[1]);
}

// Find slow handlers
function findSlowHandlers(thresholdMs = 100) {
  const messages = debug.getTrace();
  return messages.filter(msg =>
    msg.processingTime && msg.processingTime > thresholdMs
  );
}
```

21.1.5 Errors

pan-debug can encounter various error conditions during tracing operations. All errors are thrown as standard JavaScript errors.

21.1.5.1 Error Conditions

Code	Cause	Resolution
`BUFFER_OVERFLOW`	Trace buffer exceeded maxBuffer limit	Increase maxBuffer, reduce sampleRate, or clear trace more frequently
`INVALID_SAMPLE_RATE`	sampleRate outside 0.0-1.0 range	Provide valid sample rate between 0.0 and 1.0
`QUERY_FAILED`	findMessages() query malformed	Check query object format (topic, clientId, timeRange)
`EXPORT_FAILED`	exportTrace() serialization error	Check for circular references or non-serializable data
`MEMORY_EXCEEDED`	Trace buffer consuming too much memory	Reduce maxBuffer or use lower sampleRate

21.1.5.2 Error Handling

```
import { PanDebugManager } from './pan-debug.mjs';

const debug = new PanDebugManager();

// Safe initialization with error handling
function initializeDebugger(options = {}) {
  try {
    // Validate sample rate
    const sampleRate = options.sampleRate ?? 1.0;
    if (sampleRate < 0 || sampleRate > 1.0) {
      throw new Error('INVALID_SAMPLE_RATE: Sample rate must be
        between 0.0 and 1.0');
    }

    // Enable tracing with validated options
    debug.enableTracing({
      maxBuffer: options.maxBuffer || 1000,
      sampleRate: sampleRate
    });

    console.log('Debug tracing enabled');
  } catch (err) {
    console.error('Debug initialization failed:', err.message);
    // Fallback to minimal tracing
    debug.enableTracing({ maxBuffer: 100, sampleRate: 0.01 });
  }
}

// Safe query with error handling
function safeQuery(queryOptions) {
  try {
    // Validate time range
```

```
    if (queryOptions.timeRange) {
      const { start, end } = queryOptions.timeRange;
      if (start && end && start > end) {
        throw new Error('QUERY_FAILED: Invalid time range (start >
        end)');
      }
    }

    const results = debug.findMessages(queryOptions);
    return results;
  } catch (err) {
    console.error('Query failed:', err.message);
    return [];
  }
}

// Safe export with size checking
function safeExport(format = 'json') {
  try {
    const stats = debug.getStats();

    // Check buffer size before export
    if (stats.tracedMessages > 10000) {
      console.warn('Large trace buffer, export may be slow');
      // Optionally export in chunks
      const chunkSize = 1000;
      const chunks = [];
      for (let i = 0; i < stats.tracedMessages; i += chunkSize) {
        const chunk = debug.getTrace({ limit: chunkSize, offset: i
        });
        chunks.push(chunk);
      }
      return chunks;
```

```
    }

    const exportData = debug.exportTrace(format);
    return exportData;
  } catch (err) {
    console.error('Export failed:', err.message);
    // Return minimal diagnostic data
    return { error: err.message, stats: debug.getStats() };
  }
}

// Memory monitoring
function monitorMemory() {
  const stats = debug.getStats();
  const estimatedSize = stats.bufferSize * 1000; // ~1KB per message

  if (estimatedSize > 10 * 1024 * 1024) { // 10MB threshold
    console.warn('MEMORY_EXCEEDED: Trace buffer exceeding 10MB');
    debug.clearTrace();
    debug.setSampleRate(0.1); // Reduce to 10%
  }
}

// Periodic memory check
setInterval(monitorMemory, 30000);

// Example: Safe debug UI with error boundaries
function buildRobustDebugPanel() {
  try {
    initializeDebugger({ maxBuffer: 500, sampleRate: 0.1 });

    const panel = document.createElement('div');
    panel.id = 'debug-panel';
```

```
  const refresh = () => {
    try {
      const messages = safeQuery({ limit: 20 });
      const stats = debug.getStats();

      panel.innerHTML = `
        <h3>PAN Debug (${stats.totalMessages} total,
      ${stats.droppedMessages} dropped)</h3>
        <p>Buffer: ${stats.tracedMessages}/${stats.bufferSize} |
      Rate: ${stats.sampleRate * 100}%</p>
        <button onclick="clearDebugTrace()">Clear</button>
        <button onclick="exportDebugTrace()">Export</button>
        ${stats.droppedMessages > 0 ? '<span style="color:red;">⚠
      Messages dropped</span>' : ''}
        <hr>
        ${messages.length === 0 ? '<p>No messages traced</p>' :
          messages.map(msg => `
            <div style="margin: 5px 0; padding: 5px; border: 1px
      solid #ddd;">
              <strong>${msg.topic}</strong>
              <small>${new
      Date(msg.timestamp).toLocaleTimeString()}</small>
              <pre>${JSON.stringify(msg.data, null,
      2).substring(0, 200)}</pre>
            </div>
          `).join('')
        }
      `;
    } catch (err) {
      panel.innerHTML = `<p style="color:red;">Error:
      ${err.message}</p>`;
    }
  };
```

```
    setInterval(refresh, 2000);
    refresh();
    document.body.appendChild(panel);

    // Global safe controls
    window.clearDebugTrace = () => {
      try {
        debug.clearTrace();
        refresh();
      } catch (err) {
        console.error('Clear failed:', err.message);
      }
    };

    window.exportDebugTrace = () => {
      try {
        const data = safeExport();
        console.log('Exported trace:', data);
        // Optionally download as file
        const blob = new Blob([JSON.stringify(data, null, 2)], {
        type: 'application/json' });
        const url = URL.createObjectURL(blob);
        const a = document.createElement('a');
        a.href = url;
        a.download = `pan-trace-${Date.now()}.json`;
        a.click();
      } catch (err) {
        console.error('Export failed:', err.message);
      }
    };

  } catch (err) {
```

```
    console.error('Debug panel initialization failed:',
        err.message);
  }
}
```

21.1.5.3 Exceptions Thrown

- **TypeError**: Invalid parameter types (e.g., non-numeric sample rate)
- **Error**: General tracing errors (buffer overflow, query failures, export errors)

21.1.5.4 Performance Considerations

- Large buffers (>1000 messages) increase memory usage significantly
- High sample rates (>0.1) can impact application performance
- Export operations on large traces (>5000 messages) may block UI thread
- Consider Web Workers for large export operations in production

21.1.6 Common Issues

High memory usage: Reduce `maxBuffer` or `sampleRate`. For production, use `sampleRate: 0.01` (1%).

Messages not appearing: Ensure tracing enabled and sampling rate > 0.

Performance impact: Tracing adds overhead. Use sampling (0.01-0.1) in production, or disable entirely.

21.2 pan-forwarder

Purpose: Forward PAN messages to HTTP endpoints
Import: `<pan-forwarder>` custom element

21.2.1 Quick Example

```
<pan-forwarder
  url="https://api.example.com/events"
  topics="user.login,user.logout,error.*"
  method="POST"
  debounce="500">
</pan-forwarder>

<script type="module">
// Messages matching topics are automatically forwarded
// No additional code needed
</script>
```

21.2.2 Attributes

Attribute	Type	Default	Description
`url`	String	“”	HTTP endpoint URL
`topics`	String	“*”	Comma-separated topic patterns
`method`	String	“POST”	HTTP method
`debounce`	Number	0	Debounce delay (ms)
`batch-size`	Number	1	Batch multiple messages
`batch-timeout`	Number	1000	Max batch wait time (ms)
`credentials`	String	“”	Fetch credentials mode
`deduplicate`	Boolean	false	Skip duplicate messages

21.2.3 Headers Configuration

Configure headers via embedded JSON script:

```
<pan-forwarder url="/api/events" topics="user.*">
  <script type="application/json">
    {
      "headers": {
        "Authorization": "Bearer TOKEN",
        "X-App-Version": "1.0.0"
      }
    }
  </script>
</pan-forwarder>
```

21.2.4 Request Format

Single message:

```
{
  "topic": "user.login",
  "data": { "userId": 123 },
  "timestamp": 1640000000000,
  "id": "msg-abc123"
}
```

Batched messages:

```
{
  "messages": [
    { "topic": "user.login", "data": {...}, "timestamp":
        1640000000000 },
    { "topic": "user.action", "data": {...}, "timestamp":
        1640000001000 }
  ],
  "batchSize": 2
}
```

21.2.5 Complete Example

```
<!DOCTYPE html>
<html>
<head>
  <script type="module">
    import './pan-bus.mjs';
    import './pan-forwarder.mjs';
    import { PanClient } from './pan-client.mjs';

    customElements.whenDefined('pan-bus').then(() => {
      const pc = new PanClient();

      // Application publishes messages normally
      document.getElementById('login-
        btn').addEventListener('click', () => {
        pc.publish('user.login', {
          userId: 123,
          timestamp: Date.now()
        });
        // Message automatically forwarded to server
      });

      // Track forwarding status
      pc.subscribe('forwarder.success', (msg) => {
        console.log('Forwarded:', msg.data.topic);
      });

      pc.subscribe('forwarder.error', (msg) => {
        console.error('Forward failed:', msg.data.error);
      });
    });
  </script>
</head>
```

```
<body>
  <pan-bus></pan-bus>

  <!-- Forward user events to analytics server -->
  <pan-forwarder
    url="https://analytics.example.com/events"
    topics="user.login,user.logout,user.action"
    method="POST"
    debounce="1000"
    batch-size="10"
    batch-timeout="5000">
    <script type="application/json">
      {
        "headers": {
          "Authorization": "Bearer analytics-token",
          "Content-Type": "application/json"
        }
      }
    </script>
  </pan-forwarder>

  <!-- Forward errors to monitoring service -->
  <pan-forwarder
    url="https://monitoring.example.com/errors"
    topics="error.*,*.error"
    method="POST"
    credentials="include">
    <script type="application/json">
      {
        "headers": {
          "X-Service": "my-app",
          "X-Environment": "production"
        }
```

```
      }
    </script>
  </pan-forwarder>

  <button id="login-btn">Login (forwards to server)</button>
</body>
</html>
```

21.2.6 Server-Side Example

Node.js/Express:

```
app.post('/events', express.json(), (req, res) => {
  const { topic, data, timestamp } = req.body;

  console.log(`Received: ${topic}`, data);

  // Process event
  if (topic === 'user.login') {
    analytics.track('login', data);
  }

  res.json({ success: true });
});

// Batched messages
app.post('/events/batch', express.json(), (req, res) => {
  const { messages } = req.body;

  messages.forEach(msg => {
    console.log(`Received: ${msg.topic}`, msg.data);
  });

  res.json({ success: true, processed: messages.length });
});
```

21.2.7 PAN Events

21.2.7.1 Published

Topic	Data	Description
`forwarder.success`	`{ topic, url, status }`	Message forwarded successfully
`forwarder.error`	`{ topic, url, error, status }`	Forward failed

21.2.8 Errors

pan-forwarder can encounter various error conditions when forwarding messages to HTTP endpoints. All errors are published as PAN messages on the `forwarder.error` topic.

21.2.8.1 Error Conditions

Code	Cause	Resolution
`NETWORK_ERROR`	Network connection failed	Check internet connectivity, verify URL is reachable
`HTTP_ERROR`	Server returned error status (4xx, 5xx)	Check server logs, verify authentication, check endpoint exists
`BATCH_TIMEOUT`	Batch timeout exceeded before filling	Reduce batch-timeout or batch-size for low-volume scenarios
`TOPIC_MISMATCH`	No topics matched messages	Verify topic patterns, use “*” to match all messages
`PARSE_ERROR`	Response body parsing failed	Check server returns valid JSON, verify Content-Type header
`CORS_ERROR`	Cross-origin request blocked	Configure CORS headers on server, check credentials mode
`INVALID_CONFIG`	Malformed configuration JSON	Validate JSON syntax in embedded script tag

21.2.8.2 Error Handling

```
import { PanClient } from './pan-client.mjs';

const pc = new PanClient();

// Monitor forwarding errors
pc.subscribe('forwarder.error', (msg) => {
  const { topic, url, error, status, response } = msg.data;
  console.error(`Forward failed [${topic}]:`, error);

  // Handle different error scenarios
  switch (true) {
    case error.includes('NETWORK_ERROR'):
      handleNetworkError(topic, msg.data.originalMessage);
      break;

    case status === 401:
      console.error('Authentication required');
      refreshAuthToken();
      break;

    case status === 429:
      console.warn('Rate limited, backing off');
      increaseDebounce();
      break;

    case status >= 500:
      console.warn('Server error, will retry');
      queueForRetry(topic, msg.data.originalMessage);
      break;

    case error.includes('CORS_ERROR'):
      console.error('CORS configuration issue on server');
```

```
      showCorsWarning();
      break;

    default:
      console.error('Unhandled forward error:', error);
      logToErrorTracking({ topic, error, status });
  }
});

// Track successful forwards
let forwardCount = 0;
pc.subscribe('forwarder.success', (msg) => {
  forwardCount++;
  console.log(`Forwarded ${msg.data.topic} (total:
      ${forwardCount})`);
});

// Network error handling with offline queue
const offlineQueue = [];
let isOnline = navigator.onLine;

function handleNetworkError(topic, originalMessage) {
  if (!isOnline) {
    console.log('Queueing message for offline:', topic);
    offlineQueue.push({ topic, data: originalMessage });
    localStorage.setItem('forwarder_queue',
        JSON.stringify(offlineQueue));
  }
}

// Restore queue when online
window.addEventListener('online', () => {
  isOnline = true;
```

```
  const saved = localStorage.getItem('forwarder_queue');
  if (saved) {
    const queue = JSON.parse(saved);
    console.log(`Replaying ${queue.length} queued messages`);
    queue.forEach(({ topic, data }) => {
      pc.publish(topic, data);
    });
    localStorage.removeItem('forwarder_queue');
    offlineQueue.length = 0;
  }
});

window.addEventListener('offline', () => {
  isOnline = false;
  console.warn('Network offline, queueing messages');
});

// Retry logic with exponential backoff
const retryQueue = [];
let retryTimer = null;

function queueForRetry(topic, message, retries = 0) {
  if (retries >= 3) {
    console.error('Max retries exceeded for:', topic);
    logToErrorTracking({ topic, message, error:
        'max_retries_exceeded' });
    return;
  }

  retryQueue.push({ topic, message, retries });

  if (!retryTimer) {
    const delay = Math.pow(2, retries) * 1000; // 1s, 2s, 4s
```

```
    retryTimer = setTimeout(() => {
      console.log(`Retrying ${retryQueue.length} messages...`);
      const batch = [...retryQueue];
      retryQueue.length = 0;
      retryTimer = null;

      batch.forEach(({ topic, message, retries }) => {
        pc.publish(topic, message);
        // Monitor if it fails again
        const unsubscribe = pc.subscribe('forwarder.error', (msg)
        => {
          if (msg.data.topic === topic) {
            queueForRetry(topic, message, retries + 1);
            unsubscribe();
          }
        });
      });
    }, delay);
  }
}

// Dynamic debounce adjustment for rate limiting
let currentDebounce = 0;

function increaseDebounce() {
  currentDebounce = Math.min(currentDebounce + 500, 5000); // Max 5s
  console.log(`Increased debounce to ${currentDebounce}ms`);

  // Update forwarder element
  const forwarder = document.querySelector('pan-forwarder');
  if (forwarder) {
    forwarder.setAttribute('debounce', currentDebounce);
  }
```

```
}

// Reset debounce after successful forwards
let successCount = 0;
pc.subscribe('forwarder.success', () => {
  successCount++;
  if (successCount >= 10 && currentDebounce > 0) {
    currentDebounce = Math.max(currentDebounce - 500, 0);
    console.log(`Reduced debounce to ${currentDebounce}ms`);
    const forwarder = document.querySelector('pan-forwarder');
    if (forwarder) {
      forwarder.setAttribute('debounce', currentDebounce);
    }
    successCount = 0;
  }
});

// Authentication token refresh
async function refreshAuthToken() {
  try {
    const response = await fetch('/api/refresh-token', {
      method: 'POST',
      credentials: 'include'
    });

    if (response.ok) {
      const { token } = await response.json();

      // Update forwarder headers
      const forwarder = document.querySelector('pan-forwarder');
      const scriptTag =
        forwarder.querySelector('script[type="application/json"]');
      const config = JSON.parse(scriptTag.textContent);
```

```
      config.headers.Authorization = `Bearer ${token}`;
      scriptTag.textContent = JSON.stringify(config, null, 2);

      console.log('Auth token refreshed');
    } else {
      console.error('Token refresh failed, redirecting to login');
      window.location.href = '/login';
    }
  } catch (err) {
    console.error('Token refresh error:', err.message);
  }
}

// CORS warning display
function showCorsWarning() {
  const warning = document.createElement('div');
  warning.style.cssText = `
    position: fixed; top: 20px; right: 20px;
    background: #ff9800; color: white; padding: 15px;
    border-radius: 5px; box-shadow: 0 2px 10px rgba(0,0,0,0.2);
    z-index: 10000;
  `;
  warning.innerHTML = `
    <strong>⚠ CORS Error</strong><br>
    Server needs CORS headers configured.<br>
    <button onclick="this.parentElement.remove()">Dismiss</button>
  `;
  document.body.appendChild(warning);
}

// Error tracking integration
function logToErrorTracking(errorData) {
  // Example: Send to monitoring service
```

```
  fetch('https://monitoring.example.com/errors', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      component: 'pan-forwarder',
      timestamp: Date.now(),
      ...errorData
    })
  }).catch(err => console.error('Error tracking failed:',
        err.message));
}

// Example: Robust forwarder with comprehensive error handling
function setupRobustForwarder() {
  const container = document.createElement('div');
  container.innerHTML = `
    <pan-forwarder
      url="https://api.example.com/events"
      topics="user.*,error.*"
      method="POST"
      debounce="1000"
      batch-size="10"
      batch-timeout="5000"
      credentials="include">
      <script type="application/json">
        {
          "headers": {
            "Authorization": "Bearer initial-token",
            "Content-Type": "application/json",
            "X-Client-Version": "1.0.0"
          }
        }
      </script>
```

```
    </pan-forwarder>

    <div id="forward-stats" style="position: fixed; bottom: 10px;
        right: 10px; background: white; padding: 10px; border: 1px
        solid #ddd; font-family: monospace; font-size: 12px;">
      Forwards: <span id="success-count">0</span> |
      Errors: <span id="error-count">0</span> |
      Queued: <span id="queue-count">0</span>
    </div>
  `;

  document.body.appendChild(container);

  // Update stats display
  let errorCount = 0;
  pc.subscribe('forwarder.error', () => {
    errorCount++;
    document.getElementById('error-count').textContent = errorCount;
    document.getElementById('queue-count').textContent =
        offlineQueue.length + retryQueue.length;
  });

  pc.subscribe('forwarder.success', () => {
    document.getElementById('success-count').textContent =
        forwardCount;
  });

  // Periodic queue check
  setInterval(() => {
    document.getElementById('queue-count').textContent =
        offlineQueue.length + retryQueue.length;
  }, 1000);
}
```

```
// Initialize when ready
customElements.whenDefined('pan-bus').then(setupRobustForwarder);
```

21.2.8.3 Exceptions Thrown

pan-forwarder does not throw exceptions directly. All errors are published as PAN messages on `forwarder.error` topic.

21.2.8.4 HTTP Status Codes

Common status codes encountered:

- **401 Unauthorized**: Missing or invalid authentication token
- **403 Forbidden**: Valid token but insufficient permissions
- **404 Not Found**: Endpoint URL incorrect
- **429 Too Many Requests**: Rate limiting in effect
- **500 Internal Server Error**: Server-side processing error
- **502 Bad Gateway**: Server unreachable or down
- **503 Service Unavailable**: Server temporarily overloaded

21.2.8.5 Server Requirements

For successful message forwarding, server must:

1. Accept POST requests (or configured method) at the URL
2. Support JSON request body (Content-Type: application/json)
3. Return JSON response with status field
4. Configure CORS headers if cross-origin:

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: POST, OPTIONS
Access-Control-Allow-Headers: Content-Type, Authorization
```

5. Handle batched messages if batch-size > 1
6. Respond within reasonable timeout (default: 30s)

21.2.9 Common Issues

Messages not forwarded: Check topic patterns match. Use `topics="*"` to forward all messages.

CORS errors: Configure CORS headers on server. Use `credentials="include"` for cookies.

Too many requests: Increase `debounce` or use `batch-size` to group messages.

Duplicate forwards: Enable `deduplicate="true"` to skip duplicate messages within 5-second window.

Server overload: Use batching (`batch-size="50"` , `batch-timeout="5000"`) to reduce request frequency.

21.3 Summary

This chapter documented LARC's utility components:

- **pan-debug**: Message tracing and debugging with query capabilities
- **pan-forwarder**: HTTP message forwarding for server integration

These components provide observability and extensibility without modifying core

application logic.

See Also:

- Tutorial: *Learning LARC* Chapter 14
- Core components: Chapter 17
- Integration patterns: Chapter 20
- Debugging guide: Appendix D

22 Message Topics Reference

This appendix provides a comprehensive reference for LARC message topic conventions. Topics are the fundamental addressing mechanism in LARC applications—they determine how messages are routed, who receives them, and how components interact. Understanding topic patterns is essential for building scalable, maintainable LARC applications.

22.1 Topic Format and Structure

22.1.1 Standard Format

LARC topics follow a hierarchical dotted notation:

```
resource.action.qualifier
```

Components:

- **Resource:** The domain entity or system (users, posts, nav, ui, auth)
- **Action:** The operation or event type (list, item, get, save, updated)
- **Qualifier:** Additional context (state, request, reply, event)

This three-segment format provides clarity while remaining flexible enough for various use cases. More segments can be added when needed for specificity.

22.1.2 Topic Examples

Topic	Resource	Action	Qualifier	Purpose
`users.list.state`	users	list	state	Current user list (retained)
`users.item.get`	users	item	get	Fetch single user (request)
`posts.item.updated`	posts	item	updated	Post was modified (event)
`nav.goto`	nav	goto	—	Navigate to route (command)
`ui.modal.opened`	ui	modal.opened	—	Modal opened (event)
`auth.session.state`	auth	session	state	Current session (retained)

22.1.3 Naming Rules

Case and Separators:

- Use lowercase letters only
- Separate segments with dots (`.`)
- Use hyphens (`-`) within a segment if needed: `auth.two-factor.verify`

Character Set: - Alphanumeric characters: `a-z` , `0-9` - Dots for segment separation: `.` - Hyphens for compound words: `-` - No underscores, spaces, or special characters

Length: - Keep topics concise but descriptive - Typical range: 15-40 characters - Avoid abbreviations that sacrifice clarity

22.2 Topic Patterns and Matching

22.2.1 Exact Match

The simplest pattern matches a specific topic exactly:

```
client.subscribe('users.item.updated', (msg) => {
  console.log('User updated:', msg.data);
});
```

Receives only messages published to `users.item.updated` .

22.2.2 Single-Segment Wildcard

The asterisk (`*`) matches exactly one segment:

Pattern	Matches	Does Not Match
`users.*`	`users.list`, `users.item`	`users.list.state`, `users.item.updated`
`*.updated`	`users.updated`, `posts.updated`	`users.item.updated`
`users.*.state`	`users.list.state`, `users.item.state`	`users.state`, `users.list.item.state`

```
// Subscribe to all user-related events
client.subscribe('users.*', (msg) => {
  console.log('User event:', msg.topic);
});
```

22.2.3 Global Wildcard

The special pattern `*` matches all topics:

```
// Monitor all messages (use sparingly)
client.subscribe('*', (msg) => {
  console.log('[ALL]', msg.topic, msg.data);
});
```

Warning: Global wildcard subscriptions match every message in the system. Use them only for debugging, logging, or analytics. They can significantly impact performance in high-throughput applications.

22.2.4 Pattern Matching Examples

```
// Match all list operations
client.subscribe('*.list.*', (msg) => {
  // Matches: users.list.state, posts.list.get, comments.list.get
});

// Match all state topics
client.subscribe('*.*.state', (msg) => {
  // Matches: users.list.state, app.theme.state, nav.route.state
});

// Match all operations on items
client.subscribe('*.item.*', (msg) => {
  // Matches: users.item.get, posts.item.save, users.item.updated
});
```

22.3 Reserved Topic Namespaces

Certain topic namespaces are reserved for LARC internal use. Applications must not publish to these topics directly.

22.3.1 pan:* Namespace (System Internal)

The `pan:*` namespace is reserved for PAN bus internals:

Topic	Purpose	Usage
`pan:sys.ready`	Bus ready signal	Listen only
`pan:sys.stats`	Bus statistics	Request only
`pan:sys.error`	System errors	Listen only
`pan:sys.clear-retained`	Clear retained messages	Request only
`pan:publish`	Internal publish event	Internal only
`pan:subscribe`	Internal subscribe event	Internal only
`pan:unsubscribe`	Internal unsubscribe event	Internal only
`pan:deliver`	Internal message delivery	Internal only
`pan:hello`	Client registration	Internal only
`pan:$reply:*`	Auto-generated reply topics	Internal only

Never: - Publish to `pan:*` topics directly - Subscribe to internal topics like `pan:publish` or `pan:subscribe` - Manually create `pan:$reply:*` topics (these are auto-generated by `request()`)

Exception: You may subscribe to system notification topics like `pan:sys.ready` and `pan:sys.error`.

22.3.2 sys:* Namespace (System Reserved)

The `sys:*` namespace is reserved for future system-level functionality:

```
sys:error               # Future: System-wide errors
sys:perf                # Future: Performance monitoring
sys:debug               # Future: Debug information
sys:config             # Future: Configuration changes
```

Do not use `sys:*` topics in application code.

22.3.3 Application Namespaces

Your application should establish its own top-level namespaces:

Namespace	Purpose	Examples
`app.*`	Application-level concerns	`app.config.state`, `app.theme.state`
`auth.*`	Authentication/ authorization	`auth.login`, `auth.session.state`
`session.*`	Session management	`session.started`, `session.expired`
`nav.*`	Navigation	`nav.goto`, `nav.route.state`
`ui.*`	UI components	`ui.modal.opened`, `ui.toast.show`
`analytics.*`	Analytics tracking	`analytics.event`, `analytics.page-view`

22.4 CRUD Topic Patterns

CRUD operations (Create, Read, Update, Delete) follow consistent topic patterns across all resources.

22.4.1 List Operations

22.4.1.1 List State (Retained)

Topic: `${resource}.list.state`

Purpose: Retained snapshot of current list data. New subscribers receive the most recent list immediately.

Message Format:

```
{
  topic: 'users.list.state',
  data: {
    items: [/* array of items */],
    total: 150,              // Total count (for pagination)
    page: 1,                 // Current page
    filter: {},              // Active filters
    sort: 'name-asc'         // Active sort
  },
  retain: true
}
```

Usage:

```
// Publish list state
client.publish({
  topic: 'users.list.state',
  data: {
    items: users,
    total: users.length,
    page: 1
  },
  retain: true
});

// Subscribe to list state
client.subscribe('users.list.state', (msg) => {
  renderList(msg.data.items);
}, { retained: true });
```

22.4.1.2 List Get (Request)

Topic: `${resource}.list.get`

Purpose: Request to fetch list data with optional parameters.

Request Format:

```
{
  topic: 'users.list.get',
  data: {
    page: 1,
    limit: 20,
    filter: { active: true },
    sort: 'name-asc'
  }
}
```

Response Format:

```
{
  ok: true,
  items: [/* array */],
  total: 150,
  page: 1
}
```

Usage:

```
// Request list
const response = await client.request('users.list.get', {
  page: 1,
  limit: 20,
  filter: { active: true }
});

if (response.data.ok) {
  renderList(response.data.items);
}
```

22.4.2 Item Operations

22.4.2.1 Item Get (Request)

Topic: `${resource}.item.get`

Purpose: Fetch a single item by ID.

Request Format:

```
{
  topic: 'users.item.get',
  data: { id: 123 }
}
```

Response Format:

```
// Success
{ ok: true, item: { id: 123, name: 'Alice', email: '...' } }

// Not found
{ ok: false, error: 'Not found', code: 'NOT_FOUND' }
```

22.4.2.2 Item Save (Request)

Topic: `${resource}.item.save`

Purpose: Create or update an item. If `id` is present, updates existing item. If `id` is omitted, creates new item.

Request Format:

```
{
  topic: 'users.item.save',
  data: {
    item: {
      id: 123,              // Omit for create
      name: 'Alice',
      email: 'alice@example.com'
    }
  }
}
```

Response Format:

```
// Success
{ ok: true, item: { id: 123, name: 'Alice', email: '...' } }

// Validation error
{ ok: false, error: 'Invalid email', code: 'VALIDATION_ERROR' }
```

22.4.2.3 Item Delete (Request)

Topic: `${resource}.item.delete`

Purpose: Delete an item by ID.

Request Format:

```
{
  topic: 'users.item.delete',
  data: { id: 123 }
}
```

Response Format:

```
// Success
{ ok: true, id: 123 }

// Not found
{ ok: false, error: 'Not found', code: 'NOT_FOUND' }
```

22.4.2.4 Item Select (Event)

Topic: `${resource}.item.select`

Purpose: User selected/focused an item (no reply expected).

Message Format:

```
{
  topic: 'users.item.select',
  data: { id: 123 }
}
```

Usage:

```
// Publish selection
client.publish({
  topic: 'users.item.select',
  data: { id: userId }
});

// Handle selection
client.subscribe('users.item.select', (msg) => {
  highlightItem(msg.data.id);
  loadDetails(msg.data.id);
});
```

22.4.3 Item Events

Item events notify about completed operations:

Topic	Trigger	Data
`\${resource}.item.created`	Item created	`{ item: {...} }`
`\${resource}.item.updated`	Item updated	`{ item: {...} }`
`\${resource}.item.deleted`	Item deleted	`{ id: 123 }`

Example:

```
// After save operation completes
client.publish({
  topic: 'users.item.updated',
  data: { item: savedUser }
});
```

22.4.4 Per-Item State

For tracking state of individual items:

Topic: `${resource}.item.state.${id}`

Purpose: Retained state for a specific item (e.g., online status, typing indicator).

Example:

```
// Publish item state
client.publish({
  topic: `users.item.state.${userId}`,
  data: {
    id: userId,
    online: true,
    typing: false,
    lastSeen: Date.now()
  },
  retain: true
});

// Subscribe to specific item
client.subscribe(`users.item.state.${userId}`, (msg) => {
  updatePresence(msg.data);
}, { retained: true });

// Subscribe to all item states
client.subscribe('users.item.state.*', (msg) => {
  updatePresence(msg.data);
});
```

22.5 State Management Topics

State topics use the `.state` qualifier and are always retained.

22.5.1 Global State

Pattern: `${domain}.state`

Examples:

```
'app.config.state'          # Application configuration
'app.theme.state'            # Current theme
'app.language.state'         # Current language
'ui.sidebar.state'           # Sidebar open/closed
'ui.loading.state'           # Loading indicator
```

Usage:

```
// Publish state
client.publish({
  topic: 'app.theme.state',
  data: { mode: 'dark', accent: '#007bff' },
  retain: true
});

// Subscribe (receives current state immediately)
client.subscribe('app.theme.state', (msg) => {
  applyTheme(msg.data);
}, { retained: true });
```

22.5.2 Scoped State

Pattern: `${domain}.${scope}.state`

Examples:

```
'users.list.state'          # User list
'auth.session.state'        # Current session
'nav.route.state'           # Current route
'search.query.state'        # Search query
'filters.active.state'     # Active filters
```

22.6 Events vs Commands

Distinguish between events (past tense) and commands (imperative).

22.6.1 Events

Events describe something that **already happened**. They use past tense.

Characteristics: - Past tense verbs: `created`, `updated`, `deleted`, `opened`, `closed` - Fire-and-forget (no reply expected) - Multiple subscribers allowed - Informational

Examples:

Topic	Description
`users.item.created`	A user was created
`users.item.updated`	A user was updated
`ui.modal.opened`	A modal was opened
`ui.modal.closed`	A modal was closed
`session.started`	Session started
`session.expired`	Session expired
`auth.login.success`	Login succeeded
`auth.login.failed`	Login failed
`nav.navigated`	Navigation completed

Usage:

```
// Publish event
client.publish({
  topic: 'users.item.created',
  data: { item: newUser }
});

// Multiple handlers can react
client.subscribe('users.item.created', logAnalytics);
client.subscribe('users.item.created', sendWelcomeEmail);
client.subscribe('users.item.created', updateDashboard);
```

22.6.2 Commands

Commands request something to **happen**. They use imperative/verb form.

Characteristics: - Imperative verbs: `save`, `delete`, `open`, `close`, `goto` - May expect reply (request/reply pattern) - Usually single handler - May fail

Examples:

Topic	Description
`users.item.save`	Save a user
`users.item.delete`	Delete a user
`ui.modal.open`	Open a modal
`ui.modal.close`	Close a modal
`nav.goto`	Navigate to route
`nav.back`	Go back in history
`auth.login`	Perform login
`auth.logout`	Perform logout

Usage:

```
// Fire-and-forget command
client.publish({
  topic: 'nav.goto',
  data: { route: '/users/123' }
});

// Request command (expect reply)
const response = await client.request('users.item.save', {
  item: { name: 'Alice' }
});
```

22.7 Domain-Specific Patterns

22.7.1 Authentication

```
// Commands
'auth.login'                # Login request
'auth.logout'               # Logout request
'auth.refresh'              # Refresh token
'auth.verify'               # Verify credentials

// Events
'auth.login.success'        # Login succeeded
'auth.login.failed'         # Login failed
'auth.logout'               # User logged out
'auth.token.expired'        # Token expired

// State
'auth.session.state'        # Current session (retained)
'auth.user.state'           # Current user info (retained)
```

22.7.2 Navigation

```
// Commands
'nav.goto'                  # Navigate to route
'nav.back'                  # Go back
'nav.forward'               # Go forward
'nav.replace'               # Replace current route

// Events
'nav.navigated'             # Navigation completed
'nav.error'                 # Navigation error

// State
'nav.route.state'           # Current route (retained)
'nav.history.state'         # History stack (retained)
```

22.7.3 UI Components

```
// Modal
'ui.modal.open'             # Command: open modal
'ui.modal.close'            # Command: close modal
'ui.modal.opened'           # Event: modal opened
'ui.modal.closed'           # Event: modal closed
'ui.modal.state'            # State: current modal (retained)

// Sidebar
'ui.sidebar.toggle'         # Command: toggle sidebar
'ui.sidebar.open'           # Command: open sidebar
'ui.sidebar.close'          # Command: close sidebar
'ui.sidebar.state'          # State: open/closed (retained)

// Toast
'ui.toast.show'             # Command: show toast
'ui.toast.hide'             # Command: hide toast

// Loading
'ui.loading.start'          # Command: start loading
'ui.loading.stop'           # Command: stop loading
'ui.loading.state'          # State: loading status (retained)
```

22.7.4 Forms

```
// Validation
'form.validate'              # Command: validate form
'form.validated'             # Event: validation complete
'form.validation.state'      # State: validation errors (retained)

// Submission
'form.submit'                # Command: submit form
'form.submitted'             # Event: form submitted
'form.submit.success'        # Event: submission succeeded
'form.submit.failed'         # Event: submission failed

// Field changes
'form.field.changed'        # Event: field value changed
'form.field.focused'        # Event: field focused
'form.field.blurred'        # Event: field blurred
```

22.7.5 Data Synchronization

```
// Sync commands
'sync.start'                # Start sync
'sync.stop'                 # Stop sync
'sync.refresh'              # Force refresh

// Sync events
'sync.started'              # Sync started
'sync.completed'            # Sync completed
'sync.failed'               # Sync failed
'sync.conflict'            # Sync conflict detected

// Sync state
'sync.status.state'         # Current sync status (retained)
'sync.last-update.state'    # Last update time (retained)
```

22.8 Best Practices

22.8.1 Topic Naming

DO: - Use lowercase letters - Use dots to separate segments - Use descriptive names - Be consistent across resources - Use `.state` for retained topics - Use past tense for events - Use imperative for commands

DON'T: - Use underscores or camelCase - Use abbreviations that sacrifice clarity - Mix naming conventions - Use verbs for events (`users.update` X -> `users.updated` [check]) - Overuse wildcards (`*` matches everything)

22.8.2 Topic Catalog

For larger applications, maintain a centralized topic catalog:

```
// topics.js
export const TOPICS = {
  USERS: {
    LIST: {
      STATE: 'users.list.state',
      GET: 'users.list.get'
    },
    ITEM: {
      GET: 'users.item.get',
      SAVE: 'users.item.save',
      DELETE: 'users.item.delete',
      SELECT: 'users.item.select',
      UPDATED: 'users.item.updated',
      DELETED: 'users.item.deleted',
      STATE: (id) => `users.item.state.${id}`
    }
  },

  NAV: {
    GOTO: 'nav.goto',
    BACK: 'nav.back',
    ROUTE_STATE: 'nav.route.state'
  },

  AUTH: {
    LOGIN: 'auth.login',
    LOGOUT: 'auth.logout',
    SESSION_STATE: 'auth.session.state'
  }
};

// Usage
```

```
client.publish({
  topic: TOPICS.USERS.ITEM.UPDATED,
  data: { item: user }
});
```

22.8.3 Performance Considerations

Wildcard Usage: - Avoid global wildcard (`*`) in production code - Prefer specific patterns (`users.*` over `*`) - Each wildcard increases matching overhead

Topic Depth: - Keep topics shallow (3-4 segments ideal) - Deeper hierarchies increase matching cost - Balance specificity with performance

State Retention: - Use retention sparingly (only for actual state) - Don't retain high-volume event streams - Clear retained messages when no longer needed

22.9 Summary

Key Principles:

1. **Standard Format:** `resource.action.qualifier`
2. **State Suffix:** Always use `.state` for retained topics
3. **Events vs Commands:** Past tense for events, imperative for commands
4. **Consistency:** Use same patterns across all resources
5. **Reserved Namespaces:** Never use `pan:*` or `sys:*`
6. **Wildcards:** Use judiciously; prefer specific patterns
7. **Documentation:** Maintain topic catalog for large apps

Quick Reference:

Pattern	Example	Use Case
`\${resource}.list.state`	`users.list.state`	List data (retained)
`\${resource}.list.get`	`users.list.get`	Request list
`\${resource}.item.get`	`users.item.get`	Request single item
`\${resource}.item.save`	`users.item.save`	Save item
`\${resource}.item.delete`	`users.item.delete`	Delete item
`\${resource}.item.updated`	`users.item.updated`	Item updated event
`\${domain}.state`	`app.theme.state`	Global state (retained)
`\${domain}.\${action}`	`nav.goto`	Command

For complete API documentation, see the main API Reference.

23 Event Envelope Specification

This appendix provides the complete specification for LARC message envelopes—the data structures that wrap every message flowing through the PAN bus. Understanding the envelope format is critical for debugging, building tooling, and understanding how the system works at a fundamental level.

23.1 Overview

Every message in LARC is wrapped in an envelope that provides metadata, routing information, and payload data. The envelope follows a simple, predictable structure that balances flexibility with consistency.

Key Characteristics:

- Plain JavaScript objects (no classes or prototypes)
- JSON-serializable (can be logged, stored, transmitted)
- Immutable once published (bus may add fields, but won't modify existing)
- Extensible through headers and custom fields

23.2 Message Envelope Structure

23.2.1 Complete Format

```
interface PanMessage {
  // Required fields (must be provided by publisher)
  topic: string;
  data: any;

  // Auto-generated fields (added by bus if not provided)
  id?: string;
  ts?: number;

  // Optional feature fields
  retain?: boolean;
  replyTo?: string;
  correlationId?: string;
  headers?: Record<string, string>;

  // Internal/system fields (typically not used by applications)
  clientId?: string;
}
```

23.2.2 Minimal Message

The absolute minimum required to publish a message:

```
{
  topic: 'users.updated',
  data: { id: 123, name: 'Alice' }
}
```

The bus will enhance this to:

```
{
  topic: 'users.updated',
  data: { id: 123, name: 'Alice' },
  id: '550e8400-e29b-41d4-a716-446655440000',
  ts: 1699564800000
}
```

23.3 Field Specifications

23.3.1 topic (required)

Type: `string`

Purpose: Identifies the message type and routing destination.

Format: Dotted notation, typically `resource.action.qualifier`

Constraints: - Must be non-empty string - Lowercase letters and dots recommended - Max length: 256 characters (practical limit) - Pattern: `/^[a-z0-9.-]+$/i`

Examples:

```
'users.list.state'
'users.item.get'
'nav.goto'
'ui.modal.opened'
'auth.session.state'
```

Validation:

```
// Valid topics
'users.updated'              [v]
'nav.goto'                   [v]
'users.item.state.123'       [v]
'auth.two-factor.verify'     [v]

// Invalid topics
''                           [x] Empty string
'users updated'              [x] Contains space
'users_updated'              [x] Underscore (not recommended)
null                         [x] Not a string
```

Reserved Patterns:

- `pan:*` - Reserved for PAN bus internals
- `sys:*` - Reserved for system-level topics

23.3.2 data (required)

Type: `any` (must be JSON-serializable)

Purpose: Message payload—the actual information being communicated.

Constraints: - Must be JSON-serializable (no functions, circular refs, DOM nodes) -

Recommended max size: 512KB (configurable via `max-payload-size`) - Can be any valid JSON type: object, array, string, number, boolean, null

Supported Types:

```
// Object
{ id: 123, name: 'Alice', active: true }

// Array
[{ id: 1 }, { id: 2 }, { id: 3 }]

// String
"Hello, world"

// Number
42
3.14159

// Boolean
true
false

// Null
null
```

Invalid Data:

```
// Functions
data: () => console.log('hi')    [x]

// undefined (use null instead)
data: undefined                  [x]

// Circular references
const obj = {};
obj.self = obj;
data: obj                        [x]

// DOM nodes
data: document.body              [x]
```

Best Practices:

```
// Good: structured data
{
  topic: 'users.item.updated',
  data: {
    id: 123,
    name: 'Alice',
    email: 'alice@example.com',
    updatedAt: Date.now()
  }
}

// Good: minimal data
{
  topic: 'users.item.select',
  data: { id: 123 }
}

// Good: null for no data
{
  topic: 'ui.modal.close',
  data: null
}

// Acceptable: primitive data
{
  topic: 'counter.value',
  data: 42
}

// Bad: empty object when null is better
{
  topic: 'ui.modal.close',
```

```
  data: {}  // Use null instead
}
```

23.3.3 id (optional, auto-generated)

Type: `string` (UUID v4)

Purpose: Unique identifier for message deduplication, tracking, and correlation.

Auto-generation: If not provided, bus generates a UUID v4.

Format: Standard UUID format: `xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx`

Example:

```
'550e8400-e29b-41d4-a716-446655440000'
'7c9e6679-7425-40de-944b-e07fc1f90ae7'
```

Usage:

```
// Let bus generate (recommended)
client.publish({
  topic: 'users.updated',
  data: { id: 123 }
  // id will be auto-generated
});

// Provide custom ID (rare)
client.publish({
  topic: 'users.updated',
  data: { id: 123 },
  id: 'user-update-123-2024-11-01'
});
```

Use Cases: - Deduplication (detect duplicate messages) - Message tracking in logs - Correlation across systems - Idempotency keys

23.3.4 ts (optional, auto-generated)

Type: `number` (Unix timestamp in milliseconds)

Purpose: Message creation timestamp for ordering and time-based filtering.

Auto-generation: If not provided, bus adds `Date.now()`.

Format: Milliseconds since Unix epoch (January 1, 1970 00:00:00 UTC)

Example:

```
1699564800000  // 2023-11-10 00:00:00 UTC
1699651200000  // 2023-11-11 00:00:00 UTC
```

Usage:

```
// Let bus generate (recommended)
client.publish({
  topic: 'users.updated',
  data: { id: 123 }
  // ts will be auto-generated
});

// Provide custom timestamp (rare)
client.publish({
  topic: 'users.updated',
  data: { id: 123 },
  ts: Date.parse('2024-11-01T12:00:00Z')
});
```

Use Cases: - Message ordering - Time-based filtering - Analytics and logging - Determining message freshness

Working with Timestamps:

```
// Convert to Date object
const date = new Date(msg.ts);

// Format for display
const formatted = new Date(msg.ts).toISOString();
// "2024-11-01T12:00:00.000Z"

// Check message age
const ageMs = Date.now() - msg.ts;
const ageSeconds = ageMs / 1000;
```

23.3.5 retain (optional)

Type: `boolean`

Purpose: Indicates message should be retained by bus and replayed to new subscribers.

Default: `false`

Constraints: - Only one message retained per topic (last value wins) - Subject to LRU eviction if bus retention limit exceeded - Bus default max retained: 1000 messages (configurable via `max-retained`)

Usage:

```
// Publish retained state
client.publish({
  topic: 'app.theme.state',
  data: { mode: 'dark' },
  retain: true
});

// Later subscriber receives immediately
client.subscribe('app.theme.state', (msg) => {
  applyTheme(msg.data);
}, { retained: true });
```

When to Use: - Application state (theme, language, configuration) - List data (current user list, current items) - Session information (current user, authentication) - Last known values (device status, connection state)

When NOT to Use: - Events (one-time notifications) - High-frequency updates (mouse movements, scroll events) - Temporary notifications (toasts, alerts) - Bulk data (large lists, file contents)

23.3.6 replyTo (optional)

Type: `string` (topic name)

Purpose: Specifies topic where reply should be sent (request/reply pattern).

Auto-generation: Auto-generated by `client.request()` method.

Format: Typically `pan:$reply:${clientId}:${correlationId}`

Usage:

```
// Manually set replyTo
client.publish({
  topic: 'users.item.get',
  data: { id: 123 },
  replyTo: 'users.item.get.reply.abc123',
  correlationId: 'req-001'
});

// Subscribe to reply
client.subscribe('users.item.get.reply.abc123', (msg) => {
  console.log('Response:', msg.data);
});

// Better: use client.request() (auto-generates replyTo)
const response = await client.request('users.item.get', { id: 123
        });
```

Responder Pattern:

```
client.subscribe('users.item.get', async (msg) => {
  // Check if reply expected
  if (!msg.replyTo) return;

  const user = await database.getUser(msg.data.id);

  // Send reply to specified topic
  client.publish({
    topic: msg.replyTo,
    data: user ? { ok: true, item: user } : { ok: false, error: 'Not
        found' },
    correlationId: msg.correlationId
  });
});
```

23.3.7 correlationId (optional)

Type: `string` (UUID or custom identifier)

Purpose: Correlates replies with original requests in request/reply pattern.

Auto-generation: Auto-generated by `client.request()` method.

Format: Typically UUID, but can be any string identifier.

Usage:

```
// Manual correlation (rare)
const corrId = crypto.randomUUID();

client.publish({
  topic: 'users.item.get',
  data: { id: 123 },
  replyTo: 'users.reply',
  correlationId: corrId
});

client.subscribe('users.reply', (msg) => {
  if (msg.correlationId === corrId) {
    console.log('Our reply:', msg.data);
  }
});

// Better: use client.request() (auto-correlates)
const response = await client.request('users.item.get', { id: 123
        });
```

Use Cases: - Request/reply pattern - Tracking conversation threads - Matching async responses - Distributed tracing

23.3.8 headers (optional)

Type: `Record<string, string>` (string key-value pairs)

Purpose: Free-form metadata for custom application needs.

Constraints: - Keys and values must be strings - No reserved header names (yet) - Included in message size calculations

Common Use Cases:

- User context (userId, sessionId, tenantId)
- Distributed tracing (traceId, spanId, parentSpanId)
- Source tracking (source, version, environment)
- Custom metadata (priority, category, tags)

Usage:

```
// Add headers
client.publish({
  topic: 'analytics.event',
  data: { action: 'click', target: 'button' },
  headers: {
    userId: '123',
    sessionId: 'abc-def-ghi',
    source: 'mobile-app',
    version: '2.1.0',
    environment: 'production'
  }
});

// Access headers in subscriber
client.subscribe('analytics.*', (msg) => {
  const userId = msg.headers?.userId;
  const source = msg.headers?.source;

  logEvent(msg.topic, msg.data, { userId, source });
});
```

Distributed Tracing Example:

```
// Start trace
const traceId = crypto.randomUUID();
const spanId = crypto.randomUUID();

client.publish({
  topic: 'users.item.get',
  data: { id: 123 },
  headers: {
    traceId,
    spanId,
    parentSpanId: null
  }
});

// Continue trace in handler
client.subscribe('users.item.get', async (msg) => {
  const childSpanId = crypto.randomUUID();

  // Process with trace context
  await database.getUser(msg.data.id, {
    traceId: msg.headers.traceId,
    parentSpanId: msg.headers.spanId,
    spanId: childSpanId
  });
});
```

23.3.9 clientId (internal)

Type: `string`

Purpose: Internal identifier for client that published the message.

Auto-generation: Generated by PanClient constructor.

Format: `${elementTag}#${uuid}`

Example: `pan-user-list#7c9e6679-7425-40de-944b`

Usage: Primarily for internal bus operations. Applications typically don't need to use this field.

23.4 Message Examples

23.4.1 Simple Event

```
{
  topic: 'users.item.updated',
  data: {
    id: 123,
    name: 'Alice Cooper',
    email: 'alice@example.com',
    updatedAt: 1699564800000
  },
  id: '550e8400-e29b-41d4-a716-446655440000',
  ts: 1699564800000
}
```

23.4.2 Retained State

```
{
  topic: 'users.list.state',
  data: {
    items: [
      { id: 1, name: 'Alice' },
      { id: 2, name: 'Bob' },
      { id: 3, name: 'Charlie' }
    ],
    total: 3,
    page: 1
  },
  id: '7c9e6679-7425-40de-944b-e07fc1f90ae7',
  ts: 1699651200000,
  retain: true
}
```

23.4.3 Request Message

```
{
  topic: 'users.item.get',
  data: { id: 123 },
  id: 'a1b2c3d4-e5f6-4a5b-8c9d-0e1f2a3b4c5d',
  ts: 1699651200000,
  replyTo: 'pan:$reply:user-list#abc123:req-001',
  correlationId: 'req-001'
}
```

23.4.4 Reply Message

```
{
  topic: 'pan:$reply:user-list#abc123:req-001',
  data: {
    ok: true,
    item: {
      id: 123,
      name: 'Alice',
      email: 'alice@example.com'
    }
  },
  id: 'b2c3d4e5-f6a7-4b5c-9d0e-1f2a3b4c5d6e',
  ts: 1699651205000,
  correlationId: 'req-001'
}
```

23.4.5 Message with Headers

```
{
  topic: 'analytics.page-view',
  data: {
    page: '/users/123',
    duration: 5000,
    referrer: '/dashboard'
  },
  id: 'c3d4e5f6-a7b8-4c5d-0e1f-2a3b4c5d6e7f',
  ts: 1699651200000,
  headers: {
    userId: '456',
    sessionId: 'session-xyz',
    device: 'mobile',
    browser: 'Chrome',
    version: '119.0',
    traceId: 'trace-abc-def'
  }
}
```

23.4.6 Error Response

```
{
  topic: 'pan:$reply:user-form#xyz789:req-002',
  data: {
    ok: false,
    error: 'User not found',
    code: 'NOT_FOUND',
    details: {
      requestedId: 999,
      timestamp: 1699651200000
    }
  },
  id: 'd4e5f6a7-b8c9-4d5e-1f2a-3b4c5d6e7f8a',
  ts: 1699651210000,
  correlationId: 'req-002'
}
```

23.5 Response Payload Conventions

While `data` can be any JSON value, LARC follows conventions for response payloads in request/reply patterns:

23.5.1 Success Response

```
{
  ok: true,
  item: { /* single item */ }
}

// or for lists
{
  ok: true,
  items: [ /* array of items */ ],
  total: 100
}
```

23.5.2 Error Response

```
{
  ok: false,
  error: 'Human-readable error message',
  code: 'ERROR_CODE',           // Optional: machine-readable code
  details: { /* context */ }     // Optional: additional context
}
```

23.5.3 Example Error Codes

```
'NOT_FOUND'           // Resource doesn't exist
'VALIDATION_ERROR'    // Invalid input
'UNAUTHORIZED'        // Not authenticated
'FORBIDDEN'           // Not authorized
'CONFLICT'            // Resource conflict (e.g., duplicate)
'RATE_LIMIT'          // Too many requests
'SERVER_ERROR'        // Internal error
'TIMEOUT'             // Operation timed out
```

23.6 Message Size Limits

The PAN bus enforces configurable size limits:

23.6.1 Default Limits

- **Max message size:** 1MB (1,048,576 bytes)
- **Max payload size:** 512KB (524,288 bytes)

23.6.2 Configuration

```
<pan-bus
  max-message-size="2097152"
  max-payload-size="1048576">
</pan-bus>
```

23.6.3 Size Estimation

The bus estimates message size by JSON-stringifying and measuring:

```
function estimateSize(msg) {
  return new Blob([JSON.stringify(msg)]).size;
}
```

23.6.4 Handling Size Limits

```
// Large data: paginate requests
const response = await client.request('users.list.get', {
  page: 1,
  limit: 50  // Smaller page size
});

// Large payloads: use chunking or external storage
// Instead of:
client.publish({
  topic: 'file.uploaded',
  data: { fileContents: hugeBase64String }  // Too large!
});

// Do:
const fileId = await uploadToStorage(fileContents);
client.publish({
  topic: 'file.uploaded',
  data: { fileId, url: `/files/${fileId}` }
});
```

23.7 Validation

The PAN bus validates messages before processing:

23.7.1 Topic Validation

```
// Must be non-empty string
topic: ''                    [x]
topic: null                  [x]
topic: undefined             [x]

// Must not be reserved
topic: 'pan:publish'         [x] (reserved)
topic: 'pan:subscribe'       [x] (reserved)

// Valid
topic: 'users.updated'       [v]
```

23.7.2 Data Validation

```
// Must be JSON-serializable
data: () => {}                [x] (function)
data: document.body           [x] (DOM node)
data: undefined              [x] (use null)

const obj = {};
obj.self = obj;
data: obj                     [x] (circular reference)

// Valid
data: null                    [v]
data: { id: 123 }             [v]
data: [1, 2, 3]               [v]
data: "string"                [v]
data: 42                      [v]
```

23.7.3 Size Validation

Messages exceeding size limits are rejected with error:

```
// Error emitted if message too large
{
  topic: 'pan:sys.error',
  data: {
    code: 'MESSAGE_INVALID',
    message: 'Message size (2000000 bytes) exceeds limit (1048576
        bytes)',
    details: { topic: 'users.list.state' }
  }
}
```

23.8 Internal System Messages

Certain system messages are emitted by the bus:

23.8.1 pan:sys.ready

Emitted when bus initializes:

```
{
  topic: 'pan:sys.ready',
  data: {
    enhanced: true,
    routing: false,
    tracing: false,
    config: { /* bus configuration */ }
  }
}
```

23.8.2 pan:sys.error

Emitted for validation errors, rate limits, etc:

```
{
  topic: 'pan:sys.error',
  data: {
    code: 'RATE_LIMIT_EXCEEDED',
    message: 'Too many messages',
    details: { clientId: 'user-list#abc' }
  }
}
```

23.8.3 pan:sys.stats

Response to stats request:

```
{
  topic: 'pan:sys.stats',
  data: {
    published: 1234,
    delivered: 5678,
    dropped: 0,
    retainedEvicted: 5,
    subsCleanedUp: 2,
    errors: 1,
    subscriptions: 18,
    clients: 5,
    retained: 42,
    config: { /* current config */ }
  }
}
```

23.9 Summary

Required Fields: - `topic` (string) - Message routing address - `data` (any) - JSON-serializable payload

Auto-Generated Fields:

- `id` (string) - UUID for deduplication/tracking
- `ts` (number) - Unix timestamp in milliseconds

Feature Fields: - `retain` (boolean) - Retain for late subscribers - `replyTo` (string) - Reply destination topic - `correlationId` (string) - Request/reply correlation - `headers` (object) - Custom metadata

Constraints: - Topic: non-empty string, max 256 chars - Data: JSON-serializable, max 512KB (configurable) - Total message: max 1MB (configurable) - Headers: string key-

value pairs only

Best Practices: - Let bus auto-generate `id` and `ts` - Use structured objects for `data` - Use `retain: true` only for actual state - Follow response conventions (`ok`, `error`, `code`) - Include relevant context in `headers` for tracing - Keep messages small; paginate or reference external storage

For topic naming conventions, see Appendix A. For configuration options, see Appendix C.

24 Configuration Options

This appendix provides a comprehensive reference for all configuration options available in LARC, from PAN bus settings to component configuration, global defaults, and environment variables. These settings control performance characteristics, security policies, feature flags, and operational behavior.

24.1 PAN Bus Configuration

The `<pan-bus>` element accepts configuration through HTML attributes. These settings control the bus's behavior, resource limits, and enabled features.

24.1.1 Attribute Reference

24.1.1.1 max-retained

Type: Integer **Default:** `1000` **Range:** `1` to `100000` **Purpose:** Maximum number of retained messages stored by the bus.

When this limit is exceeded, the bus evicts the least recently accessed retained message (LRU eviction).

```
<pan-bus max-retained="5000"></pan-bus>
```

Use Cases: - Small apps with minimal state: `100-500` - Medium apps: `1000-2000` (default: `1000`) - Large apps with extensive state: `5000-10000`

Performance Impact:

- Higher values: More memory usage, slower eviction checks
- Lower values: Less memory, faster eviction, more message loss

Monitoring:

```
const stats = await client.request('pan:sys.stats', {});
console.log('Retained:', stats.data.retained);
console.log('Evicted:', stats.data.retainedEvicted);
```

24.1.1.2 max-message-size

Type: Integer **Default:** `1048576` (1MB) **Range:** `1024` to `10485760` (1KB to 10MB) **Purpose:** Maximum total size of a message envelope in bytes.

Includes all fields: topic, data, headers, id, ts, etc.

```
<pan-bus max-message-size="2097152"></pan-bus>
```

Recommendations: - Default `1048576` (1MB) suitable for most apps - Increase for apps with large payloads (analytics, file metadata) - Decrease for memory-constrained environments (IoT, embedded)

Enforcement:

```
// Message rejected if too large
{
  topic: 'pan:sys.error',
  data: {
    code: 'MESSAGE_INVALID',
    message: 'Message size (2000000 bytes) exceeds limit (1048576
        bytes)'
  }
}
```

24.1.1.3 max-payload-size

Type: Integer **Default:** `524288` (512KB) **Range:** `1024` to `5242880` (1KB to 5MB) **Purpose:** Maximum size of message `data` field in bytes.

Separate limit for payload to prevent large data objects from consuming resources.

```
<pan-bus max-payload-size="1048576"></pan-bus>
```

Relationship to max-message-size:

```
max-payload-size <= max-message-size
```

Best Practice: Set `max-payload-size` to 50-70% of `max-message-size` to leave room for headers and metadata.

24.1.1.4 cleanup-interval

Type: Integer **Default:** `30000` (30 seconds) **Range:** `1000` to `300000` (1s to 5min) **Purpose:** Interval in milliseconds between automatic cleanup cycles.

The bus periodically removes dead subscriptions (components removed from DOM) and stale rate limit data.

```
<pan-bus cleanup-interval="60000"></pan-bus>
```

Tuning: - Frequent cleanup (10-20s): Lower memory, higher CPU - Infrequent cleanup (60-120s): Higher memory, lower CPU - Default (30s): Balanced

Monitoring:

```
const stats = await client.request('pan:sys.stats', {});
console.log('Cleaned up:', stats.data.subsCleanedUp);
```

24.1.1.5 rate-limit

Type: Integer **Default:** `1000` **Range:** `1` to `100000` **Purpose:** Maximum messages per client per rate limit window.

Prevents a single client from overwhelming the bus.

```
<pan-bus rate-limit="5000"></pan-bus>
```

Calculation:

```
messages_per_second = rate-limit / (rate-limit-window / 1000)

Default: 1000 / (1000 / 1000) = 1000 messages/second
```

Rate Limit Exceeded:

```
// Message rejected
{
  topic: 'pan:sys.error',
  data: {
    code: 'RATE_LIMIT_EXCEEDED',
    message: 'Too many messages',
    details: { clientId: 'user-list#abc123' }
  }
}
```

Recommendations: - Development: `1000-5000` (relaxed) - Production: `1000-2000` (default) - High-throughput apps: `5000-10000`

24.1.1.6 rate-limit-window

Type: Integer **Default:** `1000` (1 second) **Range:** `100` to `60000` (100ms to 1min) **Purpose:** Time window in milliseconds for rate limiting.

Works together with `rate-limit` to determine messages per time window.

```
<pan-bus rate-limit="2000" rate-limit-window="2000"></pan-bus>
```

This allows 2000 messages per 2 seconds = 1000 messages/second.

24.1.1.7 allow-global-wildcard

Type: Boolean **Default:** `true` **Purpose:** Allow or disallow global wildcard (`*`) subscriptions.

Global wildcard subscriptions match every message in the system. Disabling can improve security and performance.

```
<pan-bus allow-global-wildcard="false"></pan-bus>
```

When Disabled:

```
// Subscription rejected
client.subscribe('*', handler);
// Error: Global wildcard (*) subscriptions are disabled for
        security
```

Recommendations: - Development: `true` (useful for debugging) - Production: `false` (security/performance)

24.1.1.8 debug

Type: Boolean **Default:** `false` **Purpose:** Enable detailed console logging of bus operations.

```
<pan-bus debug="true"></pan-bus>
```

Output Examples:

```
[PAN Bus] PAN Bus Enhanced ready { maxRetained: 1000, ... }
[PAN Bus] Client registered { id: 'user-list#abc', caps: ['client']
}
[PAN Bus] Subscription added { pattern: 'users.*', clientId: 'user-
list#abc' }
[PAN Bus] Published { topic: 'users.updated', delivered: 3, routes:
0 }
[PAN Bus] Cleaned up 2 dead subscriptions
```

Performance Impact: Minimal (logging is cheap), but avoid in production.

24.1.1.9 enable-routing

Type: Boolean **Default:** `false` **Purpose:** Enable the advanced routing system for message transformation and filtering.

```
<pan-bus enable-routing="true"></pan-bus>
```

Features Enabled:

- Declarative routing rules
- Message transformation
- Message filtering
- Topic aliasing
- Conditional routing

Access Routing API:

```
// Routes available at window.pan.routes
window.pan.routes.add({
  name: 'user-events-to-analytics',
  match: 'users.item.*',
  transform: (msg) => ({
    topic: 'analytics.event',
    data: { entity: 'user', action: msg.topic, ...msg.data }
  })
});
```

Performance Impact: Slight overhead per message for route matching.

24.1.1.10 enable-tracing

Type: Boolean **Default:** `false` **Purpose:** Enable message tracing for debugging and monitoring.

```
<pan-bus enable-tracing="true"></pan-bus>
```

Features Enabled:

- Message flow visualization
- Delivery tracking
- Route matching logs
- Performance metrics

Access Tracing API:

```
// Debug manager available at window.pan.debug
const traces = window.pan.debug.getTraces();
console.log('Recent traces:', traces);
```

24.1.2 Complete Configuration Example

```
<!DOCTYPE html>
<html>
<head>
  <title>My LARC App</title>
</head>
<body>
  <pan-bus
    max-retained="2000"
    max-message-size="2097152"
    max-payload-size="1048576"
    cleanup-interval="60000"
    rate-limit="5000"
    rate-limit-window="1000"
    allow-global-wildcard="false"
    debug="false"
    enable-routing="false"
    enable-tracing="false">
  </pan-bus>

  <my-app></my-app>
</body>
</html>
```

24.2 PanClient Configuration

The `PanClient` class accepts configuration through constructor parameters and method options.

24.2.1 Constructor Options

```
new PanClient(host?, busSelector?)
```

24.2.1.1 host

Type: `HTMLElement | Document` **Default:** `document` **Purpose:** Element to dispatch/receive events from.

```
// Default: document-level client
const client = new PanClient();

// Component-scoped client
class MyComponent extends HTMLElement {
  connectedCallback() {
    this.client = new PanClient(this);
  }
}
```

Use Cases: - Document-level (`document`): Most common, global communication - Component-scoped (element): Isolated communication within subtree

24.2.1.2 busSelector

Type: `string` **Default:** `'pan-bus'` **Purpose:** CSS selector for bus element.

```
// Default selector
const client = new PanClient(document, 'pan-bus');

// Custom selector
const client = new PanClient(document, '#my-custom-bus');
```

Rarely needed unless using multiple buses or custom naming.

24.2.2 Subscription Options

```
client.subscribe(topics, handler, options?)
```

24.2.2.1 retained

Type: `boolean` **Default:** `false` **Purpose:** Receive retained messages immediately upon subscription.

```
// Get current state immediately
client.subscribe('app.theme.state', (msg) => {
  applyTheme(msg.data);
}, { retained: true });
```

Behavior: - `true`: Receives retained message immediately (if exists), then future messages - `false`: Only receives messages published after subscription

24.2.2.2 signal

Type: `AbortSignal` **Default:** `undefined` **Purpose:** Automatically unsubscribe when signal is aborted.

```
const controller = new AbortController();

client.subscribe('users.*', handler, {
  signal: controller.signal
});

// Later: unsubscribe all at once
controller.abort();
```

Use Case: Clean up multiple subscriptions with single abort.

24.2.3 Request Options

```
client.request(topic, data, options?)
```

24.2.3.1 timeoutMs

Type: `number` **Default:** `5000` (5 seconds) **Range:** `100` to `300000` (100ms to 5min)
Purpose: Maximum time to wait for reply before timeout error.

```
// Default timeout
const response = await client.request('users.item.get', { id: 123
        });

// Custom timeout
const response = await client.request('slow.operation', { ... }, {
  timeoutMs: 30000  // 30 seconds
});
```

Timeout Error:

```
try {
  await client.request('users.item.get', { id: 123 }, { timeoutMs:
        1000 });
} catch (err) {
  console.error(err.message);  // "PAN request timeout"
}
```

Recommendations: - Fast queries: `1000-2000ms` - Standard requests: `5000ms` (default) - Slow operations: `10000-30000ms` - Long-running tasks: Consider async pattern instead

24.3 Component Configuration

LARC components can be configured through attributes, properties, and data attributes.

24.3.1 Standard Attributes

Most LARC components follow these conventions:

24.3.1.1 data-* Attributes

Use for configuration and initial values:

```
<pan-user-list
  data-page-size="50"
  data-sort="name-asc"
  data-filter="active">
</pan-user-list>
```

24.3.1.2 Boolean Attributes

Use presence/absence for boolean flags:

```
<!-- Feature enabled -->
<pan-data-grid sortable filterable paginated></pan-data-grid>

<!-- Feature disabled -->
<pan-data-grid></pan-data-grid>
```

24.3.1.3 Value Attributes

Use for string/number values:

```
<pan-modal
  size="large"
  position="center"
  backdrop="true">
</pan-modal>
```

24.3.2 Component-Specific Configuration

Configuration varies by component. Refer to component documentation for specifics.

Example: pan-storage

```
<pan-storage
  key="my-app-state"
  storage="localStorage"
  sync="true"
  debounce="1000">
</pan-storage>
```

Example: pan-routes

```
<pan-routes
  base-path="/app"
  hash-routing="false"
  scroll-restoration="true">
</pan-routes>
```

24.4 Global Configuration

Global configuration affects all LARC components and clients.

24.4.1 Window Configuration

Configure LARC globally before bus initialization:

```
<script>
window.LARC_CONFIG = {
  bus: {
    maxRetained: 2000,
    debug: false
  },
  defaults: {
    requestTimeout: 10000,
    retainedSubscription: true
  }
};
</script>

<pan-bus></pan-bus>
```

Note: This is a proposed pattern. Current implementation uses attributes only.

24.4.2 Feature Flags

Control experimental or optional features:

```
window.LARC_FEATURES = {
  routing: false,
  tracing: false,
  devtools: true
};
```

24.5 Environment Variables

For build-time configuration, use environment variables:

24.5.1 NODE_ENV

Values: `development` | `production` | `test` **Purpose:** Affects default behavior and optimizations.

```
NODE_ENV=production npm run build
```

Impact: - `development`: Debug logging, dev warnings, relaxed limits - `production`: Optimized, no debug output, strict limits - `test`: Test-specific behavior, mocked services

24.5.2 LARC_DEBUG

Values: `true` | `false` **Purpose:** Override debug mode regardless of NODE_ENV.

```
LARC_DEBUG=true npm start
```

24.5.3 LARC_MAX_RETAINED

Values: Integer **Purpose:** Override default max retained messages.

```
LARC_MAX_RETAINED=5000 npm start
```

24.5.4 LARC_RATE_LIMIT

Values: Integer **Purpose:** Override default rate limit.

```
LARC_RATE_LIMIT=10000 npm start
```

24.6 Configuration Profiles

Recommended configuration profiles for different scenarios:

24.6.1 Development Profile

Focus: Developer experience, debugging, relaxed limits

```
<pan-bus
  max-retained="500"
  max-message-size="1048576"
  max-payload-size="524288"
  cleanup-interval="30000"
  rate-limit="10000"
  allow-global-wildcard="true"
  debug="true"
  enable-routing="false"
  enable-tracing="true">
</pan-bus>
```

24.6.2 Production Profile

Focus: Performance, security, resource limits

```
<pan-bus
  max-retained="2000"
  max-message-size="1048576"
  max-payload-size="524288"
  cleanup-interval="60000"
  rate-limit="2000"
  allow-global-wildcard="false"
  debug="false"
  enable-routing="false"
  enable-tracing="false">
</pan-bus>
```

24.6.3 High-Throughput Profile

Focus: Maximum message volume, relaxed limits

```
<pan-bus
  max-retained="5000"
  max-message-size="2097152"
  max-payload-size="1048576"
  cleanup-interval="120000"
  rate-limit="10000"
  rate-limit-window="1000"
  allow-global-wildcard="false"
  debug="false"
  enable-routing="true"
  enable-tracing="false">
</pan-bus>
```

24.6.4 Memory-Constrained Profile

Focus: Minimal memory footprint

```
<pan-bus
  max-retained="100"
  max-message-size="262144"
  max-payload-size="131072"
  cleanup-interval="15000"
  rate-limit="500"
  allow-global-wildcard="false"
  debug="false"
  enable-routing="false"
  enable-tracing="false">
</pan-bus>
```

24.6.5 Testing Profile

Focus: Predictable behavior, fast cleanup

```
<pan-bus
  max-retained="50"
  max-message-size="1048576"
  max-payload-size="524288"
  cleanup-interval="5000"
  rate-limit="1000"
  allow-global-wildcard="true"
  debug="true"
  enable-routing="false"
  enable-tracing="true">
</pan-bus>
```

24.7 Runtime Configuration

Some settings can be changed at runtime through the bus API.

24.7.1 Get Current Configuration

```
const stats = await client.request('pan:sys.stats', {});
console.log('Config:', stats.data.config);
```

24.7.2 Clear Retained Messages

```
// Clear all retained messages
client.publish({
  topic: 'pan:sys.clear-retained',
  data: {}
});

// Clear matching pattern
client.publish({
  topic: 'pan:sys.clear-retained',
  data: { pattern: 'users.*' }
});
```

24.7.3 Get Statistics

```
const stats = await client.request('pan:sys.stats', {});

console.log({
  published: stats.data.published,
  delivered: stats.data.delivered,
  dropped: stats.data.dropped,
  retained: stats.data.retained,
  retainedEvicted: stats.data.retainedEvicted,
  subsCleanedUp: stats.data.subsCleanedUp,
  errors: stats.data.errors,
  subscriptions: stats.data.subscriptions,
  clients: stats.data.clients
});
```

24.8 Configuration Best Practices

24.8.1 Start Conservative

Begin with default settings:

```
<pan-bus></pan-bus>
```

Monitor with stats, adjust only when needed.

24.8.2 Monitor and Tune

```
// Periodic monitoring
setInterval(async () => {
  const stats = await client.request('pan:sys.stats', {});

  console.log('Bus Health:', {
    retained: `${stats.data.retained} /
        ${stats.data.config.maxRetained}`,
    dropped: stats.data.dropped,
    errors: stats.data.errors
  });

  // Alert if approaching limits
  if (stats.data.retained > stats.data.config.maxRetained * 0.8) {
    console.warn('Retained messages approaching limit');
  }
}, 60000);
```

24.8.3 Environment-Specific Configuration

Use different profiles per environment:

```
// config.js
export function getBusConfig() {
  if (process.env.NODE_ENV === 'production') {
    return {
      maxRetained: 2000,
      debug: false,
      rateLimit: 2000
    };
  }

  return {
    maxRetained: 500,
    debug: true,
    rateLimit: 10000
  };
}
```

```
<script type="module">
import { getBusConfig } from './config.js';

const config = getBusConfig();

document.querySelector('pan-bus').setAttribute('max-retained',
        config.maxRetained);
document.querySelector('pan-bus').setAttribute('debug',
        config.debug);
document.querySelector('pan-bus').setAttribute('rate-limit',
        config.rateLimit);
</script>
```

24.8.4 Document Your Configuration

```
<!--
  PAN Bus Configuration

  Environment: Production
  Profile: High-availability

  max-retained: 2000
    - Expected state topics: ~500
    - Headroom: 4x expected

  rate-limit: 2000
    - Expected peak load: 1000 msg/s
    - Headroom: 2x peak

  Last tuned: 2024-11-01
  Next review: 2024-12-01
-->
<pan-bus
  max-retained="2000"
  rate-limit="2000"
  debug="false">
</pan-bus>
```

24.9 Configuration Checklist

Pre-Launch: - [ ] `debug="false"` in production - [ ] `allow-global-wildcard="false"` for security - [ ] `max-retained` appropriate for app state volume - [ ] `rate-limit` appropriate for expected load - [ ] Monitoring enabled for bus

statistics - [ ] Configuration documented in code comments

Performance Tuning:

- [ ] Monitor `retained` approaching `maxRetained`
- [ ] Monitor `dropped` messages (rate limiting)
- [ ] Monitor `errors` (validation, size limits)
- [ ] Monitor `subsCleanedUp` (memory leaks?)
- [ ] Adjust limits based on observed usage

Security: - [ ] Global wildcard disabled in production - [ ] Rate limits prevent DoS - [ ] Message size limits prevent memory exhaustion - [ ] Debug mode disabled in production

24.10 Summary

PAN Bus Core Settings:

- `max-retained` - Retained message limit (default: 1000)
- `max-message-size` - Total message size limit (default: 1MB)
- `max-payload-size` - Payload size limit (default: 512KB)
- `cleanup-interval` - Cleanup cycle interval (default: 30s)
- `rate-limit` - Messages per window (default: 1000)
- `allow-global-wildcard` - Allow `*` subscriptions (default: true)
- `debug` - Enable debug logging (default: false)

PAN Bus Features:

- `enable-routing` - Enable routing system (default: false)
- `enable-tracing` - Enable message tracing (default: false)

PanClient Settings:

- `host` - Event dispatch element (default: document)

- `busSelector` - Bus element selector (default: 'pan-bus')
- `retained` - Receive retained messages (default: false)
- `timeoutMs` - Request timeout (default: 5000ms)

Recommended Profiles:

- Development: Debug enabled, relaxed limits
- Production: Debug disabled, strict limits
- High-throughput: Increased limits, routing enabled
- Memory-constrained: Minimal retained, frequent cleanup
- Testing: Fast cleanup, debug enabled

Monitoring: - Use `pan:sys.stats` to monitor bus health - Alert on approaching limits - Tune based on observed behavior - Document configuration decisions

For message envelope structure, see Appendix B. For topic conventions, see Appendix A.

25 Migration Guide

This appendix helps you upgrade LARC applications across versions, navigate breaking changes, and adopt new features while maintaining stability. Whether you're moving from an early prototype to a production release or keeping pace with framework evolution, this guide provides version-specific migration paths and practical strategies.

25.1 General Migration Strategy

Before diving into version-specific changes, establish a methodical upgrade process:

1. Review the Changelog Start with LARC's release notes. Note breaking changes, deprecations, and new features relevant to your application.

2. Update in Increments Avoid jumping multiple major versions. Upgrade one major version at a time, testing thoroughly between steps.

3. Run Your Test Suite Execute all tests before and after migration. Pay special attention to component integration tests that exercise PAN bus communication.

4. Check Dependencies Ensure your LARC-compatible libraries (routing, state management) support the new version. Update these incrementally.

5. Use Feature Flags When migrating large applications, use feature flags to toggle between old and new implementations during transition periods.

25.2 Version 0.x to 1.0 Migration

The move to LARC 1.0 established core APIs and stabilized component architecture. Key changes:

25.2.1 Component Registration Changes

Before (0.x):

```
LARC.register('my-widget', {
  template: '<div>Content</div>',
  props: ['data']
});
```

After (1.0):

```
class MyWidget extends HTMLElement {
  static observedAttributes = ['data'];

  connectedCallback() {
    this.render();
  }

  render() {
    this.innerHTML = '<div>Content</div>';
  }
}

customElements.define('my-widget', MyWidget);
```

Migration Steps: 1. Convert registration objects to ES6 classes extending `HTMLElement` 2. Move lifecycle hooks to standard Web Components callbacks 3. Implement `observedAttributes` for reactive properties 4. Replace template strings with `render()` methods

25.2.2 PAN Bus API Refinement

Version 1.0 introduced explicit bus references rather than implicit global access.

Before (0.x):

```
this.emit('data-changed', { value: 42 });
this.on('user-action', handler);
```

After (1.0):

```
this.pan.dispatch('data-changed', { value: 42 });
this.pan.subscribe('user-action', handler);
```

Migration Steps: 1. Replace `emit()` with `pan.dispatch()` 2. Replace `on()` with `pan.subscribe()` 3. Update cleanup to use returned unsubscribe functions 4. Add explicit PAN bus initialization if using custom buses

25.2.3 Attribute Handling

Attribute parsing became more strict in 1.0.

Before (0.x):

```
// Automatic JSON parsing
this.getAttribute('config'); // returns object
```

After (1.0):

```
// Explicit parsing required
JSON.parse(this.getAttribute('config') || '{}');
```

Migration Steps: 1. Add explicit JSON parsing for complex attributes 2. Implement `attributeChangedCallback()` for reactive updates 3. Use `observedAttributes` to declare monitored attributes

25.3 Version 1.x to 2.0 Migration

LARC 2.0 introduced TypeScript support, improved developer experience, and performance optimizations.

25.3.1 TypeScript Integration

While JavaScript remains fully supported, TypeScript brings type safety.

Migration Steps: 1. Install TypeScript definitions: `npm install --save-dev @larc/types` 2. Rename `.js` files to `.ts` incrementally 3. Add type annotations to component properties 4. Define custom event payload types

Example:

```
import { PANEvent } from '@larc/core';

interface UserData {
  id: string;
  name: string;
}

class UserCard extends HTMLElement {
  private userData: UserData | null = null;

  connectedCallback() {
    this.pan.subscribe<UserData>('user-selected', (event:
        PANEvent<UserData>) => {
      this.userData = event.detail;
      this.render();
    });
  }
}
```

25.3.2 Shadow DOM Adoption

Version 2.0 encouraged Shadow DOM for style encapsulation.

Before (1.x):

```
connectedCallback() {
  this.innerHTML = '<div class="container">Content</div>';
}
```

After (2.0):

```
connectedCallback() {
  this.attachShadow({ mode: 'open' });
  this.shadowRoot.innerHTML = `
    <style>
      .container { padding: 1rem; }
    </style>
    <div class="container">Content</div>
  `;
}
```

Migration Considerations:

- Global styles won't penetrate Shadow DOM
- Use CSS custom properties for theming
- Update selectors in tests to query shadow roots
- Consider performance impact for large component trees

25.3.3 Async Component Initialization

Version 2.0 added first-class async support.

Before (1.x):

```
connectedCallback() {
  fetch('/api/data').then(data => {
    this.data = data;
    this.render();
  });
}
```

After (2.0):

```
async connectedCallback() {
  await this.initialize();
}

async initialize() {
  try {
    this.data = await fetch('/api/data').then(r => r.json());
    this.render();
  } catch (error) {
    this.renderError(error);
  }
}
```

25.4 Version 2.x to 3.0 Migration

LARC 3.0 focused on performance, introducing reactive primitives and optimized rendering.

25.4.1 Reactive State Management

The new reactive state system replaces manual `render()` calls.

Before (2.x):

```
class Counter extends HTMLElement {
  constructor() {
    super();
    this.count = 0;
  }

  increment() {
    this.count++;
    this.render();
  }
}
```

After (3.0):

```
import { reactive } from '@larc/core';

class Counter extends HTMLElement {
  constructor() {
    super();
    this.state = reactive({ count: 0 });
    this.state.$watch(() => this.render());
  }

  increment() {
    this.state.count++; // automatically triggers render
  }
}
```

Migration Steps: 1. Wrap component state in `reactive()` 2. Set up `$watch()` for automatic rendering 3. Remove manual `render()` calls after state changes 4. Use `$batch()` for multiple simultaneous updates

25.4.2 PAN Bus Namespacing

Version 3.0 introduced event namespacing for better organization.

Before (2.x):

```
this.pan.dispatch('data-loaded', data);
this.pan.dispatch('data-error', error);
this.pan.dispatch('data-cleared');
```

After (3.0):

```
this.pan.dispatch('data:loaded', data);
this.pan.dispatch('data:error', error);
this.pan.dispatch('data:cleared');

// Subscribe to namespace
this.pan.subscribe('data:*', (event) => {
  console.log(`Data event: ${event.type}`);
});
```

25.4.3 Performance Optimizations

Version 3.0 added batched updates and render scheduling.

Manual Batching:

```
import { batch } from '@larc/core';

batch(() => {
  this.state.count++;
  this.state.name = 'Updated';
  this.state.timestamp = Date.now();
}); // Single render after all changes
```

Render Scheduling:

```
class HeavyComponent extends HTMLElement {
  render() {
    requestIdleCallback(() => {
      // Expensive rendering during idle time
      this.updateComplexUI();
    });
  }
}
```

25.5 Deprecation Timeline

25.5.1 Currently Deprecated (Remove in 4.0)

Legacy Event Syntax:

```
// Deprecated
this.pan.on('event-name', handler);

// Use instead
this.pan.subscribe('event-name', handler);
```

Global Bus Access:

```
// Deprecated
window.PAN.dispatch('event');

// Use instead
this.pan.dispatch('event');
```

Synchronous connectedCallback with async operations:

```
// Deprecated pattern
connectedCallback() {
  fetch('/data').then(d => this.data = d);
  this.render(); // Renders before data loads
}

// Preferred
async connectedCallback() {
  this.data = await fetch('/data').then(r => r.json());
  this.render();
}
```

25.5.2 Planned Deprecations (4.0+)

- Direct innerHTML manipulation (prefer template literals or JSX)
- Imperative event listener registration (favor declarative templates)
- String-based event names (transition to strongly-typed event enums)

25.6 Breaking Changes Checklist

When upgrading major versions, verify these common breaking change areas:

API Surface: - [ ] Component registration method - [ ] PAN bus method names - [ ] Event payload structure - [ ] Lifecycle callback signatures

Behavior Changes:

- [ ] Attribute parsing (automatic vs. manual)
- [ ] Default Shadow DOM usage
- [ ] Event bubbling and cancellation
- [ ] Async initialization timing

Build Process: - [ ] Bundler configuration - [ ] TypeScript compiler options - [ ] Test framework compatibility - [ ] Development server setup

Dependencies: - [ ] Peer dependency versions - [ ] Polyfill requirements - [ ] Browser compatibility targets - [ ] Third-party library compatibility

25.7 Migration Tools

25.7.1 Automated Refactoring

Use these tools to accelerate migration:

AST-Based Transforms:

```
npx @larc/migrate --from 2.x --to 3.0 src/**/*.js
```

Codemod Scripts:

```
// Example: Convert emit to dispatch
module.exports = function(fileInfo, api) {
  const j = api.jscodeshift;
  return j(fileInfo.source)
    .find(j.CallExpression, {
      callee: {
        object: { type: 'ThisExpression' },
        property: { name: 'emit' }
      }
    })
    .forEach(path => {
      path.value.callee.property.name = 'dispatch';
    })
    .toSource();
};
```

25.7.2 Manual Review Points

Automated tools can't catch everything. Manually review:

1. **Business Logic:** Ensure state changes behave identically
2. **Edge Cases:** Test error handling and boundary conditions
3. **Performance:** Profile before/after for regressions
4. **User Experience:** Verify visual consistency and interactions

25.8 Rollback Strategy

If migration causes critical issues:

1. **Revert Version:** Use git to restore previous package.json
2. **Isolate Changes:** Create feature branches for incremental updates
3. **Dual Implementation:** Run old and new code side-by-side with feature flags
4. **Gradual Rollout:** Deploy to subset of users before full migration

25.9 Getting Help

When stuck during migration:

- **Documentation:** Check version-specific upgrade guides at larc.dev/migrate
- **Community:** Ask in GitHub Discussions or Discord
- **Issue Tracker:** Search for similar migration problems
- **Support Contracts:** Enterprise users can access dedicated migration assistance

Migration is an investment in your application's future. Take time to understand changes, test thoroughly, and leverage community resources. The LARC team strives for smooth upgrade paths while continuing to evolve the framework.

26 Recipes and Patterns

This appendix provides practical, copy-paste-ready solutions for common LARC development scenarios. Each recipe demonstrates a specific technique or pattern you'll encounter when building real applications. Use these as starting points, adapting them to your specific requirements.

26.1 Recipe 1: Lazy-Loading Components

Defer component loading until needed, reducing initial bundle size.

```
class LazyLoader extends HTMLElement {
  async connectedCallback() {
    const componentName = this.getAttribute('component');
    const modulePath = this.getAttribute('module');

    try {
      await import(modulePath);
      const element = document.createElement(componentName);
      Array.from(this.attributes).forEach(attr => {
        if (attr.name !== 'component' && attr.name !== 'module') {
          element.setAttribute(attr.name, attr.value);
        }
      });
      this.replaceWith(element);
    } catch (error) {
      this.innerHTML = `<div class="error">Failed to load
        component</div>`;
      console.error('Lazy load failed:', error);
    }
  }
}

customElements.define('lazy-loader', LazyLoader);
```

Usage:

```
<lazy-loader
  component="data-table"
  module="/components/data-table.js"
  data-source="/api/users">
</lazy-loader>
```

When to Use: - Large components used infrequently - Route-based code splitting - Conditional feature loading based on user permissions

26.2 Recipe 2: Form Validation Component

Reusable form validation with real-time feedback.

```
class ValidatedForm extends HTMLElement {
  connectedCallback() {
    this.attachShadow({ mode: 'open' });
    this.validators = new Map();
    this.errors = new Map();

    this.shadowRoot.innerHTML = `
      <style>
        .field { margin-bottom: 1rem; }
        .error { color: #d32f2f; font-size: 0.875rem; margin-top:
        0.25rem; }
        .valid { border-color: #4caf50; }
        .invalid { border-color: #d32f2f; }
      </style>
      <form>
        <slot></slot>
        <div class="actions">
          <button type="submit">Submit</button>
        </div>
      </form>
    `;

    this.setupValidation();
  }

  setupValidation() {
    const form = this.shadowRoot.querySelector('form');
    const inputs = this.querySelectorAll('[data-validate]');

    inputs.forEach(input => {
      const rules = input.getAttribute('data-validate').split(',');
      this.validators.set(input, rules);
```

```
      input.addEventListener('blur', () =>
        this.validateField(input));
      input.addEventListener('input', () => {
        if (this.errors.has(input)) {
          this.validateField(input);
        }
      });
    });

    form.addEventListener('submit', (e) => {
      e.preventDefault();
      if (this.validateAll()) {
        this.handleSubmit();
      }
    });
  }

  validateField(input) {
    const rules = this.validators.get(input);
    const value = input.value.trim();
    let error = null;

    for (const rule of rules) {
      if (rule === 'required' && !value) {
        error = 'This field is required';
        break;
      }
      if (rule === 'email' && !this.isValidEmail(value)) {
        error = 'Invalid email address';
        break;
      }
      if (rule.startsWith('min:')) {
```

```
        const min = parseInt(rule.split(':')[1]);
        if (value.length < min) {
          error = `Minimum ${min} characters required`;
          break;
        }
      }
      if (rule.startsWith('max:')) {
        const max = parseInt(rule.split(':')[1]);
        if (value.length > max) {
          error = `Maximum ${max} characters allowed`;
          break;
        }
      }
    }

    this.updateFieldError(input, error);
    return !error;
  }

  updateFieldError(input, error) {
    input.classList.toggle('invalid', !!error);
    input.classList.toggle('valid', !error);

    let errorDiv = input.nextElementSibling;
    if (errorDiv && errorDiv.classList.contains('error')) {
      errorDiv.remove();
    }

    if (error) {
      this.errors.set(input, error);
      errorDiv = document.createElement('div');
      errorDiv.className = 'error';
      errorDiv.textContent = error;
```

```
      input.after(errorDiv);
    } else {
      this.errors.delete(input);
    }
  }

  validateAll() {
    let isValid = true;
    this.validators.forEach((rules, input) => {
      if (!this.validateField(input)) {
        isValid = false;
      }
    });
    return isValid;
  }

  isValidEmail(email) {
    return /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email);
  }

  handleSubmit() {
    const formData = new FormData(this.querySelector('form'));
    this.pan.dispatch('form:submitted',
        Object.fromEntries(formData));
  }
}

customElements.define('validated-form', ValidatedForm);
```

Usage:

```
<validated-form>
  <div class="field">
    <label>Email</label>
    <input type="email" name="email" data-validate="required,email">
  </div>
  <div class="field">
    <label>Password</label>
    <input type="password" name="password" data-
        validate="required,min:8">
  </div>
</validated-form>
```

26.3 Recipe 3: Infinite Scroll List

Load data progressively as user scrolls.

```
class InfiniteList extends HTMLElement {
  constructor() {
    super();
    this.page = 1;
    this.loading = false;
    this.hasMore = true;
  }

  connectedCallback() {
    this.apiEndpoint = this.getAttribute('api');
    this.setupIntersectionObserver();
    this.loadMore();
  }

  setupIntersectionObserver() {
    const sentinel = document.createElement('div');
    sentinel.className = 'scroll-sentinel';
    this.appendChild(sentinel);

    this.observer = new IntersectionObserver((entries) => {
      if (entries[0].isIntersecting && !this.loading &&
        this.hasMore) {
        this.loadMore();
      }
    }, { threshold: 0.1 });

    this.observer.observe(sentinel);
  }

  async loadMore() {
    this.loading = true;
    this.showLoadingIndicator();
```

```
    try {
      const response = await
        fetch(`${this.apiEndpoint}?page=${this.page}`);
      const data = await response.json();

      if (data.items.length === 0) {
        this.hasMore = false;
        this.hideLoadingIndicator();
        return;
      }

      this.renderItems(data.items);
      this.page++;
    } catch (error) {
      console.error('Failed to load items:', error);
      this.pan.dispatch('error', { message: 'Failed to load items'
        });
    } finally {
      this.loading = false;
      this.hideLoadingIndicator();
    }
  }

  renderItems(items) {
    const sentinel = this.querySelector('.scroll-sentinel');
    items.forEach(item => {
      const element = this.createItemElement(item);
      this.insertBefore(element, sentinel);
    });
  }

  createItemElement(item) {
```

```
    const div = document.createElement('div');
    div.className = 'list-item';
    div.innerHTML = `
      <h3>${item.title}</h3>
      <p>${item.description}</p>
    `;
    return div;
  }

  showLoadingIndicator() {
    let loader = this.querySelector('.loader');
    if (!loader) {
      loader = document.createElement('div');
      loader.className = 'loader';
      loader.textContent = 'Loading...';
      this.appendChild(loader);
    }
  }

  hideLoadingIndicator() {
    const loader = this.querySelector('.loader');
    if (loader) loader.remove();
  }

  disconnectedCallback() {
    if (this.observer) {
      this.observer.disconnect();
    }
  }
}

customElements.define('infinite-list', InfiniteList);
```

26.4 Recipe 4: Toast Notification System

Display temporary user notifications.

```
class ToastContainer extends HTMLElement {
  connectedCallback() {
    this.attachShadow({ mode: 'open' });
    this.shadowRoot.innerHTML = `
      <style>
        :host {
          position: fixed;
          top: 1rem;
          right: 1rem;
          z-index: 10000;
          display: flex;
          flex-direction: column;
          gap: 0.5rem;
          max-width: 400px;
        }
        .toast {
          padding: 1rem 1.5rem;
          border-radius: 0.5rem;
          box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
          display: flex;
          align-items: center;
          gap: 0.75rem;
          animation: slideIn 0.3s ease;
        }
        .toast.success { background: #4caf50; color: white; }
        .toast.error { background: #f44336; color: white; }
        .toast.info { background: #2196f3; color: white; }
        .toast.warning { background: #ff9800; color: white; }
        @keyframes slideIn {
          from {
            transform: translateX(100%);
            opacity: 0;
```

```
          }
          to {
            transform: translateX(0);
            opacity: 1;
          }
        }
        .close {
          margin-left: auto;
          cursor: pointer;
          font-size: 1.25rem;
          opacity: 0.8;
        }
        .close:hover { opacity: 1; }
      </style>
    `;

    this.pan.subscribe('toast:show', (event) => {
      this.showToast(event.detail);
    });
  }

  showToast({ message, type = 'info', duration = 3000 }) {
    const toast = document.createElement('div');
    toast.className = `toast ${type}`;
    toast.innerHTML = `
      <span class="message">${message}</span>
      <span class="close">&times;</span>
    `;

    toast.querySelector('.close').addEventListener('click', () => {
      this.removeToast(toast);
    });
```

```
    this.shadowRoot.appendChild(toast);

    if (duration > 0) {
      setTimeout(() => this.removeToast(toast), duration);
    }
  }

  removeToast(toast) {
    toast.style.animation = 'slideIn 0.3s ease reverse';
    setTimeout(() => toast.remove(), 300);
  }
}

customElements.define('toast-container', ToastContainer);
```

Usage:

```
// Anywhere in your app
this.pan.dispatch('toast:show', {
  message: 'Settings saved successfully',
  type: 'success',
  duration: 3000
});
```

26.5 Recipe 5: Debounced Search Input

Optimize API calls by debouncing user input.

```
class SearchInput extends HTMLElement {
  constructor() {
    super();
    this.debounceTimer = null;
    this.debounceDelay = parseInt(this.getAttribute('debounce')) ||
        300;
  }

  connectedCallback() {
    this.innerHTML = `
      <div class="search-wrapper">
        <input type="search" placeholder="Search...">
        <span class="spinner" style="display:
        none;">[hourglass]</span>
      </div>
      <div class="results"></div>
    `;

    this.input = this.querySelector('input');
    this.spinner = this.querySelector('.spinner');
    this.resultsContainer = this.querySelector('.results');

    this.input.addEventListener('input', (e) => {
      this.handleInput(e.target.value);
    });

    this.pan.subscribe('search:results', (event) => {
      this.displayResults(event.detail);
    });
  }

  handleInput(value) {
    clearTimeout(this.debounceTimer);
```

```
    if (!value.trim()) {
      this.resultsContainer.innerHTML = '';
      return;
    }

    this.showSpinner();

    this.debounceTimer = setTimeout(() => {
      this.performSearch(value);
    }, this.debounceDelay);
  }

  async performSearch(query) {
    try {
      const apiEndpoint = this.getAttribute('api');
      const response = await
        fetch(`${apiEndpoint}?q=${encodeURIComponent(query)}`);
      const results = await response.json();
      this.pan.dispatch('search:results', results);
    } catch (error) {
      console.error('Search failed:', error);
    } finally {
      this.hideSpinner();
    }
  }

  displayResults(results) {
    if (results.length === 0) {
      this.resultsContainer.innerHTML = '<div class="no-results">No
        results found</div>';
      return;
    }
```

```
    this.resultsContainer.innerHTML = results
      .map(result => `<div class="result-
        item">${result.title}</div>`)
      .join('');
  }

  showSpinner() {
    this.spinner.style.display = 'inline';
  }

  hideSpinner() {
    this.spinner.style.display = 'none';
  }
}

customElements.define('search-input', SearchInput);
```

26.6 Recipe 6: Modal Dialog

Accessible modal with focus trapping.

```
class ModalDialog extends HTMLElement {
  connectedCallback() {
    this.attachShadow({ mode: 'open' });
    this.shadowRoot.innerHTML = `
      <style>
        :host {
          display: none;
          position: fixed;
          top: 0;
          left: 0;
          right: 0;
          bottom: 0;
          z-index: 9999;
        }
        :host([open]) { display: block; }
        .backdrop {
          position: absolute;
          top: 0;
          left: 0;
          right: 0;
          bottom: 0;
          background: rgba(0, 0, 0, 0.5);
        }
        .modal {
          position: absolute;
          top: 50%;
          left: 50%;
          transform: translate(-50%, -50%);
          background: white;
          border-radius: 0.5rem;
          padding: 2rem;
          max-width: 90vw;
```

```
        max-height: 90vh;
        overflow: auto;
        box-shadow: 0 10px 25px rgba(0, 0, 0, 0.2);
      }
      .close {
        position: absolute;
        top: 1rem;
        right: 1rem;
        background: none;
        border: none;
        font-size: 1.5rem;
        cursor: pointer;
      }
    </style>
    <div class="backdrop"></div>
    <div class="modal" role="dialog" aria-modal="true">
      <button class="close" aria-label="Close">&times;</button>
      <slot></slot>
    </div>
  `;

  this.shadowRoot.querySelector('.backdrop').addEventListener('click', ()
      => this.close());

  this.shadowRoot.querySelector('.close').addEventListener('click', ()
      => this.close());



  this.pan.subscribe('modal:open', (event) => {
    if (event.detail.id === this.id) {
      this.open();
    }
  });
```

```
  }

  open() {
    this.setAttribute('open', '');
    this.previousFocus = document.activeElement;
    this.trapFocus();
    document.body.style.overflow = 'hidden';
  }

  close() {
    this.removeAttribute('open');
    document.body.style.overflow = '';
    if (this.previousFocus) {
      this.previousFocus.focus();
    }
    this.pan.dispatch('modal:closed', { id: this.id });
  }

  trapFocus() {
    const focusableElements = this.shadowRoot.querySelectorAll(
      'button, [href], input, select, textarea,
        [tabindex]:not([tabindex="-1"])'
    );
    const firstElement = focusableElements[0];
    const lastElement = focusableElements[focusableElements.length
        - 1];

    this.keydownHandler = (e) => {
      if (e.key !== 'Tab') return;

      if (e.shiftKey && document.activeElement === firstElement) {
        e.preventDefault();
        lastElement.focus();
```

```
      } else if (!e.shiftKey && document.activeElement ===
        lastElement) {
        e.preventDefault();
        firstElement.focus();
      }
    };

    this.addEventListener('keydown', this.keydownHandler);
    firstElement?.focus();
  }

  disconnectedCallback() {
    if (this.keydownHandler) {
      this.removeEventListener('keydown', this.keydownHandler);
    }
  }
}

customElements.define('modal-dialog', ModalDialog);
```

26.7 Recipe 7: State Persistence

Save and restore component state to localStorage.

```
class StatefulComponent extends HTMLElement {
  constructor() {
    super();
    this.storageKey = this.getAttribute('storage-key') ||
        'component-state';
    this.state = this.loadState();
  }

  loadState() {
    try {
      const saved = localStorage.getItem(this.storageKey);
      return saved ? JSON.parse(saved) : this.getDefaultState();
    } catch (error) {
      console.error('Failed to load state:', error);
      return this.getDefaultState();
    }
  }

  saveState() {
    try {
      localStorage.setItem(this.storageKey,
        JSON.stringify(this.state));
      this.pan.dispatch('state:saved', { key: this.storageKey });
    } catch (error) {
      console.error('Failed to save state:', error);
      this.pan.dispatch('state:error', { error: error.message });
    }
  }

  updateState(updates) {
    this.state = { ...this.state, ...updates };
    this.saveState();
    this.render();
```

```
  }

  getDefaultState() {
    return {};
  }

  clearState() {
    localStorage.removeItem(this.storageKey);
    this.state = this.getDefaultState();
    this.render();
  }
}
```

26.8 Recipe 8: Drag and Drop

Reorderable list with drag-and-drop.

```
class DraggableList extends HTMLElement {
  connectedCallback() {
    this.addEventListener('dragstart',
        this.handleDragStart.bind(this));
    this.addEventListener('dragover',
        this.handleDragOver.bind(this));
    this.addEventListener('drop', this.handleDrop.bind(this));
    this.addEventListener('dragend', this.handleDragEnd.bind(this));

    this.makeItemsDraggable();
  }

  makeItemsDraggable() {
    this.querySelectorAll('.draggable-item').forEach(item => {
      item.setAttribute('draggable', 'true');
    });
  }

  handleDragStart(e) {
    if (!e.target.classList.contains('draggable-item')) return;
    e.target.classList.add('dragging');
    e.dataTransfer.effectAllowed = 'move';
    e.dataTransfer.setData('text/html', e.target.innerHTML);
  }

  handleDragOver(e) {
    e.preventDefault();
    e.dataTransfer.dropEffect = 'move';

    const dragging = this.querySelector('.dragging');
    const afterElement = this.getDragAfterElement(e.clientY);

    if (afterElement == null) {
```

```
      this.appendChild(dragging);
    } else {
      this.insertBefore(dragging, afterElement);
    }
  }

  handleDrop(e) {
    e.stopPropagation();
    this.dispatchReorderEvent();
  }

  handleDragEnd(e) {
    e.target.classList.remove('dragging');
  }

  getDragAfterElement(y) {
    const draggableElements = [
      ...this.querySelectorAll('.draggable-item:not(.dragging)')
    ];

    return draggableElements.reduce((closest, child) => {
      const box = child.getBoundingClientRect();
      const offset = y - box.top - box.height / 2;

      if (offset < 0 && offset > closest.offset) {
        return { offset: offset, element: child };
      } else {
        return closest;
      }
    }, { offset: Number.NEGATIVE_INFINITY }).element;
  }

  dispatchReorderEvent() {
```

```
    const order = Array.from(this.querySelectorAll('.draggable-
        item'))
      .map((item, index) => ({ index, id: item.dataset.id }));
    this.pan.dispatch('list:reordered', order);
  }
}

customElements.define('draggable-list', DraggableList);
```

26.9 Recipe 9: Responsive Image

Automatically load appropriate image sizes.

```
class ResponsiveImage extends HTMLElement {
  connectedCallback() {
    this.sources = JSON.parse(this.getAttribute('sources'));
    this.alt = this.getAttribute('alt') || '';

    this.render();
    window.addEventListener('resize', () => this.handleResize());
  }

  render() {
    const src = this.selectSource();
    this.innerHTML = `<img src="${src}" alt="${this.alt}"
        loading="lazy">`;
  }

  selectSource() {
    const width = window.innerWidth;
    const sorted = Object.entries(this.sources)
      .sort(([a], [b]) => parseInt(a) - parseInt(b));

    for (const [breakpoint, url] of sorted) {
      if (width <= parseInt(breakpoint)) {
        return url;
      }
    }

    return sorted[sorted.length - 1][1];
  }

  handleResize() {
    clearTimeout(this.resizeTimer);
    this.resizeTimer = setTimeout(() => {
```

```
      const currentSrc = this.querySelector('img').src;
      const newSrc = this.selectSource();
      if (currentSrc !== newSrc) {
        this.render();
      }
    }, 250);
  }
}

customElements.define('responsive-image', ResponsiveImage);
```

Usage:

```
<responsive-image
  sources='{"480": "/img/small.jpg", "1024": "/img/medium.jpg",
        "1920": "/img/large.jpg"}'
  alt="Product photo">
</responsive-image>
```

26.10 Recipe 10: Event Bus Bridge

Bridge LARC PAN bus events to external systems.

```
class EventBridge extends HTMLElement {
  connectedCallback() {
    this.externalSystem = this.getAttribute('target');
    this.eventMap = JSON.parse(this.getAttribute('event-map') ||
        '{}');

    Object.keys(this.eventMap).forEach(panEvent => {
      this.pan.subscribe(panEvent, (event) => {
        this.bridgeEvent(panEvent, event.detail);
      });
    });
  }

  bridgeEvent(panEvent, data) {
    const externalEvent = this.eventMap[panEvent];

    switch (this.externalSystem) {
      case 'analytics':
        this.sendToAnalytics(externalEvent, data);
        break;
      case 'websocket':
        this.sendToWebSocket(externalEvent, data);
        break;
      case 'postmessage':
        this.sendToParent(externalEvent, data);
        break;
    }
  }

  sendToAnalytics(event, data) {
    if (window.gtag) {
      window.gtag('event', event, data);
```

```
    }
  }

  sendToWebSocket(event, data) {
    if (this.websocket?.readyState === WebSocket.OPEN) {
      this.websocket.send(JSON.stringify({ type: event, payload:
        data }));
    }
  }

  sendToParent(event, data) {
    window.parent.postMessage({ type: event, payload: data }, '*');
  }
}

customElements.define('event-bridge', EventBridge);
```

26.11 Recipe 11: File Upload with Progress

Multi-file upload with progress tracking and validation.

```
class FileUpload extends HTMLElement {
  connectedCallback() {
    this.attachShadow({ mode: 'open' });
    this.maxSize = parseInt(this.getAttribute('max-size')) || 5 *
        1024 * 1024; // 5MB
    this.accept = this.getAttribute('accept') || '*/*';
    this.multiple = this.hasAttribute('multiple');

    this.shadowRoot.innerHTML = `
      <style>
        .upload-area {
          border: 2px dashed #ccc;
          border-radius: 0.5rem;
          padding: 2rem;
          text-align: center;
          cursor: pointer;
          transition: border-color 0.3s;
        }
        .upload-area:hover { border-color: #2196f3; }
        .upload-area.dragover { border-color: #4caf50; background:
        #f0f8f0; }
        .file-list { margin-top: 1rem; }
        .file-item {
          display: flex;
          align-items: center;
          gap: 0.5rem;
          padding: 0.5rem;
          border: 1px solid #ddd;
          border-radius: 0.25rem;
          margin-bottom: 0.5rem;
        }
        .progress-bar {
          flex: 1;
```

```
          height: 8px;
          background: #e0e0e0;
          border-radius: 4px;
          overflow: hidden;
        }
        .progress-fill {
          height: 100%;
          background: #4caf50;
          transition: width 0.3s;
        }
        .remove-btn {
          background: #f44336;
          color: white;
          border: none;
          padding: 0.25rem 0.5rem;
          border-radius: 0.25rem;
          cursor: pointer;
        }
        .error { color: #f44336; font-size: 0.875rem; }
      </style>
      <div class="upload-area">
        <input type="file" style="display: none"
        accept="${this.accept}" ${this.multiple ? 'multiple' : ''}>
        <p>📁 Drag files here or click to browse</p>
      </div>
      <div class="file-list"></div>
    `;

    this.uploadArea = this.shadowRoot.querySelector('.upload-area');
    this.fileInput =
        this.shadowRoot.querySelector('input[type="file"]');
    this.fileList = this.shadowRoot.querySelector('.file-list');
```

```
    this.setupEvents();
  }

  setupEvents() {
    this.uploadArea.addEventListener('click', () =>
        this.fileInput.click());
    this.fileInput.addEventListener('change', (e) =>
        this.handleFiles(e.target.files));

    this.uploadArea.addEventListener('dragover', (e) => {
      e.preventDefault();
      this.uploadArea.classList.add('dragover');
    });

    this.uploadArea.addEventListener('dragleave', () => {
      this.uploadArea.classList.remove('dragover');
    });

    this.uploadArea.addEventListener('drop', (e) => {
      e.preventDefault();
      this.uploadArea.classList.remove('dragover');
      this.handleFiles(e.dataTransfer.files);
    });
  }

  handleFiles(files) {
    Array.from(files).forEach(file => {
      if (file.size > this.maxSize) {
        this.showError(`${file.name} exceeds ${this.maxSize / 1024 /
        1024}MB limit`);
        return;
      }
      this.uploadFile(file);
    });
```

```
  }

  async uploadFile(file) {
    const fileId = Date.now() + Math.random();
    const fileItem = this.createFileItem(file, fileId);
    this.fileList.appendChild(fileItem);

    const formData = new FormData();
    formData.append('file', file);

    try {
      const response = await fetch(this.getAttribute('upload-url')
        || '/api/upload', {
        method: 'POST',
        body: formData,
        headers: {
          'X-File-Name': file.name
        }
      });

      if (!response.ok) throw new Error('Upload failed');

      const result = await response.json();
      this.updateProgress(fileId, 100);
      this.pan.dispatch('file:uploaded', { file: file.name, result });
    } catch (error) {
      this.showError(`Failed to upload ${file.name}`);
      fileItem.querySelector('.progress-bar').style.display = 'none';
      fileItem.querySelector('.error').textContent = error.message;
    }
  }

  createFileItem(file, id) {
```

```
    const div = document.createElement('div');
    div.className = 'file-item';
    div.dataset.id = id;
    div.innerHTML = `
      <span>${file.name} (${(file.size / 1024).toFixed(1)}KB)</span>
      <div class="progress-bar">
        <div class="progress-fill" style="width: 0%"></div>
      </div>
      <button class="remove-btn">Remove</button>
      <div class="error"></div>
    `;

    div.querySelector('.remove-btn').addEventListener('click', ()
        => {
      div.remove();
      this.pan.dispatch('file:removed', { file: file.name });
    });

    return div;
  }

  updateProgress(fileId, percent) {
    const item = this.fileList.querySelector(`[data-id="${fileId}"]`);
    if (item) {
      const fill = item.querySelector('.progress-fill');
      fill.style.width = `${percent}%`;
    }
  }

  showError(message) {
    const error = document.createElement('div');
    error.className = 'error';
    error.textContent = message;
```

```
      this.fileList.appendChild(error);
      setTimeout(() => error.remove(), 5000);
    }
  }

  customElements.define('file-upload', FileUpload);
```

26.12 Recipe 12: Autocomplete Input

Dropdown suggestions with keyboard navigation.

```
class AutocompleteInput extends HTMLElement {
  constructor() {
    super();
    this.selectedIndex = -1;
    this.suggestions = [];
  }

  connectedCallback() {
    this.attachShadow({ mode: 'open' });
    this.shadowRoot.innerHTML = `
      <style>
        .wrapper { position: relative; }
        input {
          width: 100%;
          padding: 0.5rem;
          border: 1px solid #ccc;
          border-radius: 0.25rem;
        }
        .suggestions {
          position: absolute;
          top: 100%;
          left: 0;
          right: 0;
          background: white;
          border: 1px solid #ccc;
          border-top: none;
          max-height: 200px;
          overflow-y: auto;
          z-index: 1000;
          display: none;
        }
        .suggestions.open { display: block; }
```

```
        .suggestion {
          padding: 0.5rem;
          cursor: pointer;
        }
        .suggestion:hover, .suggestion.selected {
          background: #f0f0f0;
        }
      </style>
      <div class="wrapper">
        <input type="text"
        placeholder="${this.getAttribute('placeholder') ||
        'Search...'}">
        <div class="suggestions"></div>
      </div>
    `;

    this.input = this.shadowRoot.querySelector('input');
    this.dropdown = this.shadowRoot.querySelector('.suggestions');

    this.setupEvents();
  }

  setupEvents() {
    this.input.addEventListener('input', (e) =>
        this.handleInput(e.target.value));
    this.input.addEventListener('keydown', (e) =>
        this.handleKeydown(e));
    this.input.addEventListener('blur', () => setTimeout(() =>
        this.hideDropdown(), 200));

    this.dropdown.addEventListener('click', (e) => {
      if (e.target.classList.contains('suggestion')) {
        this.selectSuggestion(e.target.textContent);
      }
```

```
    });
  }

  async handleInput(value) {
    if (value.length < 2) {
      this.hideDropdown();
      return;
    }

    try {
      const apiUrl = this.getAttribute('api');
      const response = await
        fetch(`${apiUrl}?q=${encodeURIComponent(value)}`);
      this.suggestions = await response.json();
      this.renderSuggestions();
    } catch (error) {
      console.error('Autocomplete failed:', error);
    }
  }

  renderSuggestions() {
    if (this.suggestions.length === 0) {
      this.hideDropdown();
      return;
    }

    this.dropdown.innerHTML = this.suggestions
      .map(s => `<div class="suggestion">${s}</div>`)
      .join('');
    this.dropdown.classList.add('open');
    this.selectedIndex = -1;
  }
```

```
  handleKeydown(e) {
    const items = this.dropdown.querySelectorAll('.suggestion');

    switch (e.key) {
      case 'ArrowDown':
        e.preventDefault();
        this.selectedIndex = Math.min(this.selectedIndex + 1,
        items.length - 1);
        this.updateSelection(items);
        break;
      case 'ArrowUp':
        e.preventDefault();
        this.selectedIndex = Math.max(this.selectedIndex - 1, -1);
        this.updateSelection(items);
        break;
      case 'Enter':
        e.preventDefault();
        if (this.selectedIndex >= 0) {

         this.selectSuggestion(items[this.selectedIndex].textContent);

        }
        break;
      case 'Escape':
        this.hideDropdown();
        break;
    }
  }

  updateSelection(items) {
    items.forEach((item, i) => {
      item.classList.toggle('selected', i === this.selectedIndex);
    });
```

```
  }

  selectSuggestion(value) {
    this.input.value = value;
    this.hideDropdown();
    this.pan.dispatch('autocomplete:selected', { value });
  }

  hideDropdown() {
    this.dropdown.classList.remove('open');
    this.selectedIndex = -1;
  }
}

customElements.define('autocomplete-input', AutocompleteInput);
```

26.13 Recipe 13: Sortable Data Table

Table with sortable columns and highlighting.

```
class SortableTable extends HTMLElement {
  constructor() {
    super();
    this.sortColumn = null;
    this.sortDirection = 'asc';
  }

  connectedCallback() {
    this.data = JSON.parse(this.getAttribute('data') || '[]');
    this.columns = JSON.parse(this.getAttribute('columns') || '[]');
    this.render();
  }

  render() {
    this.innerHTML = `
      <style>
        table { width: 100%; border-collapse: collapse; }
        th {
          background: #f5f5f5;
          padding: 0.75rem;
          text-align: left;
          cursor: pointer;
          user-select: none;
        }
        th:hover { background: #e0e0e0; }
        th.sorted::after {
          content: '↑';
          margin-left: 0.5rem;
        }
        th.sorted.desc::after { content: '↓'; }
        td { padding: 0.75rem; border-top: 1px solid #ddd; }
        tr:hover { background: #f9f9f9; }
```

```
      </style>
      <table>
        <thead>
          <tr>
            ${this.columns.map(col => `
              <th data-key="${col.key}">${col.label}</th>
            `).join('')}
          </tr>
        </thead>
        <tbody>
          ${this.renderRows()}
        </tbody>
      </table>
    `;

    this.querySelectorAll('th').forEach(th => {
      th.addEventListener('click', () =>
        this.handleSort(th.dataset.key));
    });
  }

  renderRows() {
    return this.data.map(row => `
      <tr>
        ${this.columns.map(col => `<td>${row[col.key] ||
        ''}</td>`).join('')}
      </tr>
    `).join('');
  }

  handleSort(key) {
    if (this.sortColumn === key) {
      this.sortDirection = this.sortDirection === 'asc' ? 'desc' :
        'asc';
```

```
    } else {
      this.sortColumn = key;
      this.sortDirection = 'asc';
    }

    this.data.sort((a, b) => {
      const aVal = a[key];
      const bVal = b[key];
      const modifier = this.sortDirection === 'asc' ? 1 : -1;

      if (typeof aVal === 'number') {
        return (aVal - bVal) * modifier;
      }
      return String(aVal).localeCompare(String(bVal)) * modifier;
    });

    this.render();
    this.updateSortIndicator();
  }

  updateSortIndicator() {
    this.querySelectorAll('th').forEach(th => {
      th.classList.remove('sorted', 'desc');
      if (th.dataset.key === this.sortColumn) {
        th.classList.add('sorted');
        if (this.sortDirection === 'desc') {
          th.classList.add('desc');
        }
      }
    });
  }
}
```

```
customElements.define('sortable-table', SortableTable);
```

26.14 Recipe 14: Tabs Component

Accessible tabs with keyboard support.

```
class TabsComponent extends HTMLElement {
  connectedCallback() {
    this.attachShadow({ mode: 'open' });
    this.currentTab = 0;
    this.render();
    this.setupKeyboardNav();
  }

  render() {
    const tabs = Array.from(this.querySelectorAll('[slot^="tab-
        "]'));
    const panels = Array.from(this.querySelectorAll('[slot^="panel-
        "]'));

    this.shadowRoot.innerHTML = `
      <style>
        .tabs {
          display: flex;
          border-bottom: 2px solid #ddd;
          gap: 0.5rem;
        }
        .tab {
          padding: 0.75rem 1.5rem;
          background: none;
          border: none;
          border-bottom: 2px solid transparent;
          cursor: pointer;
          font-size: 1rem;
          transition: all 0.3s;
        }
        .tab:hover { background: #f5f5f5; }
        .tab[aria-selected="true"] {
          border-bottom-color: #2196f3;
```

```
        color: #2196f3;
      }
      .panel {
        padding: 1.5rem 0;
        display: none;
      }
      .panel[aria-hidden="false"] { display: block; }
    </style>
    <div class="tabs" role="tablist">
      ${tabs.map((tab, i) => `
        <button
          class="tab"
          role="tab"
          aria-selected="${i === 0}"
          aria-controls="panel-${i}"
          id="tab-${i}"
          tabindex="${i === 0 ? 0 : -1}">
          ${tab.textContent}
        </button>
      `).join('')}
    </div>
    ${panels.map((panel, i) => `
      <div
        class="panel"
        role="tabpanel"
        id="panel-${i}"
        aria-labelledby="tab-${i}"
        aria-hidden="${i !== 0}">
        <slot name="panel-${i}"></slot>
      </div>
    `).join('')}
  `;
```

```
    this.shadowRoot.querySelectorAll('.tab').forEach((tab, i) => {
      tab.addEventListener('click', () => this.selectTab(i));
    });
  }

  selectTab(index) {
    this.currentTab = index;

    this.shadowRoot.querySelectorAll('.tab').forEach((tab, i) => {
      tab.setAttribute('aria-selected', i === index);
      tab.setAttribute('tabindex', i === index ? 0 : -1);
    });

    this.shadowRoot.querySelectorAll('.panel').forEach((panel, i)
        => {
      panel.setAttribute('aria-hidden', i !== index);
    });

    this.pan.dispatch('tab:changed', { index });
  }

  setupKeyboardNav() {
    this.shadowRoot.querySelector('.tabs').addEventListener('keydown', (e)
        => {

      const tabs = this.shadowRoot.querySelectorAll('.tab');
      let newIndex = this.currentTab;

      switch (e.key) {
        case 'ArrowLeft':
          newIndex = Math.max(0, this.currentTab - 1);
          break;
        case 'ArrowRight':
```

```
          newIndex = Math.min(tabs.length - 1, this.currentTab + 1);
          break;
        case 'Home':
          newIndex = 0;
          break;
        case 'End':
          newIndex = tabs.length - 1;
          break;
        default:
          return;
      }

      e.preventDefault();
      this.selectTab(newIndex);
      tabs[newIndex].focus();
    });
  }
}

customElements.define('tabs-component', TabsComponent);
```

Usage:

```
<tabs-component>
  <span slot="tab-0">Overview</span>
  <span slot="tab-1">Details</span>
  <span slot="tab-2">Reviews</span>

  <div slot="panel-0"><p>Overview content...</p></div>
  <div slot="panel-1"><p>Details content...</p></div>
  <div slot="panel-2"><p>Reviews content...</p></div>
</tabs-component>
```

26.15 Recipe 15: Accordion Component

Expandable sections with smooth animations.

```
class AccordionComponent extends HTMLElement {
  connectedCallback() {
    this.allowMultiple = this.hasAttribute('allow-multiple');
    this.render();
  }

  render() {
    const items = Array.from(this.querySelectorAll('[slot^="item-
        "]'));

    this.innerHTML = `
      <style>
        .accordion-item {
          border: 1px solid #ddd;
          margin-bottom: 0.5rem;
          border-radius: 0.25rem;
        }
        .accordion-header {
          display: flex;
          justify-content: space-between;
          align-items: center;
          padding: 1rem;
          background: #f5f5f5;
          cursor: pointer;
          user-select: none;
        }
        .accordion-header:hover { background: #e0e0e0; }
        .accordion-icon {
          transition: transform 0.3s;
        }
        .accordion-item.open .accordion-icon {
          transform: rotate(180deg);
```

```
        }
        .accordion-content {
          max-height: 0;
          overflow: hidden;
          transition: max-height 0.3s ease;
        }
        .accordion-item.open .accordion-content {
          max-height: 500px;
        }
        .accordion-body {
          padding: 1rem;
        }
      </style>
      ${items.map((item, i) => {
        const title =
        item.querySelector('[slot="title"]')?.textContent ||
        `Section ${i + 1}`;
        const content =
        item.querySelector('[slot="content"]')?.innerHTML || '';

        return `
          <div class="accordion-item" data-index="${i}">
            <div class="accordion-header">
              <span>${title}</span>
              <span class="accordion-icon">▼</span>
            </div>
            <div class="accordion-content">
              <div class="accordion-body">${content}</div>
            </div>
          </div>
        `;
      }).join('')}
    `;
```

```
    this.querySelectorAll('.accordion-header').forEach(header => {
      header.addEventListener('click', () => {
        const item = header.parentElement;
        this.toggleItem(item);
      });
    });
  }

  toggleItem(item) {
    const isOpen = item.classList.contains('open');

    if (!this.allowMultiple) {
      this.querySelectorAll('.accordion-item').forEach(i => {
        i.classList.remove('open');
      });
    }

    if (!isOpen) {
      item.classList.add('open');
      this.pan.dispatch('accordion:opened', { index:
        item.dataset.index });
    } else {
      item.classList.remove('open');
      this.pan.dispatch('accordion:closed', { index:
        item.dataset.index });
    }
  }
}

customElements.define('accordion-component', AccordionComponent);
```

26.16 Recipe 16: Context Menu

Right-click custom menu.

```
class ContextMenu extends HTMLElement {
  connectedCallback() {
    this.attachShadow({ mode: 'open' });
    this.items = JSON.parse(this.getAttribute('items') || '[]');
    this.render();
    this.setupTriggers();
  }

  render() {
    this.shadowRoot.innerHTML = `
      <style>
        :host {
          position: fixed;
          z-index: 10000;
          display: none;
        }
        :host(.visible) { display: block; }
        .menu {
          background: white;
          border: 1px solid #ddd;
          border-radius: 0.25rem;
          box-shadow: 0 2px 10px rgba(0,0,0,0.1);
          min-width: 150px;
        }
        .menu-item {
          padding: 0.5rem 1rem;
          cursor: pointer;
          display: flex;
          align-items: center;
          gap: 0.5rem;
        }
        .menu-item:hover { background: #f5f5f5; }
```

```
        .menu-separator {
          height: 1px;
          background: #ddd;
          margin: 0.25rem 0;
        }
      </style>
      <div class="menu">
        ${this.items.map(item =>
          item.separator
            ? '<div class="menu-separator"></div>'
            : `<div class="menu-item" data-action="${item.action}">
                 ${item.icon || ''} ${item.label}
               </div>`
        ).join('')}
      </div>
    `;

    this.shadowRoot.querySelectorAll('.menu-item').forEach(item => {
      item.addEventListener('click', () => {
        this.handleAction(item.dataset.action);
        this.hide();
      });
    });
  }

  setupTriggers() {
    const target = this.getAttribute('target');
    const elements = target ? document.querySelectorAll(target) :
        [document];

    elements.forEach(el => {
      el.addEventListener('contextmenu', (e) => {
        e.preventDefault();
```

```
        this.show(e.clientX, e.clientY, el);
      });
    });

    document.addEventListener('click', () => this.hide());
  }

  show(x, y, target) {
    this.style.left = `${x}px`;
    this.style.top = `${y}px`;
    this.classList.add('visible');
    this.currentTarget = target;
  }

  hide() {
    this.classList.remove('visible');
  }

  handleAction(action) {
    this.pan.dispatch('context-menu:action', {
      action,
      target: this.currentTarget
    });
  }
}

customElements.define('context-menu', ContextMenu);
```

Usage:

```
<context-menu
  target=".list-item"
  items='[
    {"label": "Edit", "action": "edit", "icon": "✏️"},
    {"label": "Delete", "action": "delete", "icon": "🗑️"},
    {"separator": true},
    {"label": "Share", "action": "share", "icon": "🔗"}
  ]'>
</context-menu>
```

26.17 Recipe 17: Copy to Clipboard

One-click copy with visual feedback.

```
class CopyButton extends HTMLElement {
  connectedCallback() {
    this.text = this.getAttribute('text') || this.textContent;
    this.render();
  }

  render() {
    this.innerHTML = `
      <button class="copy-btn">
        <span class="icon">📋</span>
        <span class="label">Copy</span>
      </button>
      <style>
        .copy-btn {
          display: inline-flex;
          align-items: center;
          gap: 0.5rem;
          padding: 0.5rem 1rem;
          border: 1px solid #ddd;
          border-radius: 0.25rem;
          background: white;
          cursor: pointer;
          transition: all 0.3s;
        }
        .copy-btn:hover {
          background: #f5f5f5;
          border-color: #2196f3;
        }
        .copy-btn.success {
          background: #4caf50;
          color: white;
          border-color: #4caf50;
```

```
        }
      </style>
    `;

    this.button = this.querySelector('.copy-btn');
    this.button.addEventListener('click', () =>
        this.copyToClipboard());
  }

  async copyToClipboard() {
    try {
      await navigator.clipboard.writeText(this.text);
      this.showSuccess();
      this.pan.dispatch('clipboard:copied', { text: this.text });
    } catch (error) {
      console.error('Copy failed:', error);
      this.showError();
    }
  }

  showSuccess() {
    const icon = this.querySelector('.icon');
    const label = this.querySelector('.label');

    icon.textContent = '✓';
    label.textContent = 'Copied!';
    this.button.classList.add('success');

    setTimeout(() => {
      icon.textContent = '📋';
      label.textContent = 'Copy';
      this.button.classList.remove('success');
    }, 2000);
```

```
  }

  showError() {
    const label = this.querySelector('.label');
    label.textContent = 'Failed';
    setTimeout(() => {
      label.textContent = 'Copy';
    }, 2000);
  }
}

customElements.define('copy-button', CopyButton);
```

26.18 Recipe 18: Dark Mode Toggle

Theme switching with system preference detection and persistence.

```
class DarkModeToggle extends HTMLElement {
  connectedCallback() {
    this.attachShadow({ mode: 'open' });
    this.initTheme();
    this.render();
  }

  initTheme() {
    const saved = localStorage.getItem('theme');
    const systemPrefers = window.matchMedia('(prefers-color-scheme:
        dark)').matches;
    this.theme = saved || (systemPrefers ? 'dark' : 'light');
    this.applyTheme();

    // Listen for system theme changes
    window.matchMedia('(prefers-color-scheme:
        dark)').addEventListener('change', (e) => {
      if (!localStorage.getItem('theme')) {
        this.theme = e.matches ? 'dark' : 'light';
        this.applyTheme();
      }
    });
  }

  render() {
    this.shadowRoot.innerHTML = `
      <style>
        button {
          background: none;
          border: 1px solid var(--border-color, #ddd);
          border-radius: 2rem;
          padding: 0.5rem 1rem;
          cursor: pointer;
```

```
          display: flex;
          align-items: center;
          gap: 0.5rem;
          font-size: 1rem;
          transition: all 0.3s;
        }
        button:hover {
          background: var(--hover-bg, #f5f5f5);
        }
      </style>
      <button>
        <span class="icon">${this.theme === 'dark' ? '🌙' :
        '☀️'}</span>
        <span>${this.theme === 'dark' ? 'Dark' : 'Light'}</span>
      </button>
    `;

    this.shadowRoot.querySelector('button').addEventListener('click', ()
        => this.toggle());

  }

  toggle() {
    this.theme = this.theme === 'light' ? 'dark' : 'light';
    this.applyTheme();
    this.saveTheme();
    this.render();
  }

  applyTheme() {
    document.documentElement.setAttribute('data-theme', this.theme);
    document.documentElement.style.colorScheme = this.theme;
    this.pan.dispatch('theme:changed', { theme: this.theme });
```

```
  }

  saveTheme() {
    localStorage.setItem('theme', this.theme);
  }
}

customElements.define('dark-mode-toggle', DarkModeToggle);
```

CSS Variables:

```
:root {
  --bg-color: white;
  --text-color: #333;
  --border-color: #ddd;
}

[data-theme="dark"] {
  --bg-color: #1a1a1a;
  --text-color: #f0f0f0;
  --border-color: #444;
}

body {
  background: var(--bg-color);
  color: var(--text-color);
  transition: background 0.3s, color 0.3s;
}
```

26.19 Recipe 19: Skeleton Loader

Loading placeholders for better perceived performance.

```
class SkeletonLoader extends HTMLElement {
  connectedCallback() {
    this.type = this.getAttribute('type') || 'text';
    this.lines = parseInt(this.getAttribute('lines')) || 3;
    this.render();
  }

  render() {
    const templates = {
      text: this.renderTextSkeleton(),
      card: this.renderCardSkeleton(),
      list: this.renderListSkeleton(),
      profile: this.renderProfileSkeleton()
    };

    this.innerHTML = `
      <style>
        .skeleton {
          animation: pulse 1.5s ease-in-out infinite;
        }
        @keyframes pulse {
          0%, 100% { opacity: 1; }
          50% { opacity: 0.4; }
        }
        .skeleton-line {
          height: 1rem;
          background: #e0e0e0;
          border-radius: 0.25rem;
          margin-bottom: 0.5rem;
        }
        .skeleton-circle {
          width: 50px;
```

```
          height: 50px;
          border-radius: 50%;
          background: #e0e0e0;
        }
        .skeleton-rect {
          height: 200px;
          background: #e0e0e0;
          border-radius: 0.5rem;
          margin-bottom: 1rem;
        }
      </style>
      ${templates[this.type] || templates.text}
    `;
  }

  renderTextSkeleton() {
    return Array(this.lines)
      .fill(0)
      .map((_, i) => {
        const width = i === this.lines - 1 ? '60%' : '100%';
        return `<div class="skeleton skeleton-line" style="width:
        ${width}"></div>`;
      })
      .join('');
  }

  renderCardSkeleton() {
    return `
      <div class="skeleton skeleton-rect"></div>
      ${this.renderTextSkeleton()}
    `;
  }
```

```
  renderListSkeleton() {
    return Array(this.lines)
      .fill(0)
      .map(() => `
        <div style="display: flex; gap: 1rem; margin-bottom: 1rem;">
          <div class="skeleton skeleton-circle"></div>
          <div style="flex: 1;">
            <div class="skeleton skeleton-line"></div>
            <div class="skeleton skeleton-line" style="width:
          70%"></div>
          </div>
        </div>
      `)
      .join('');
  }

  renderProfileSkeleton() {
    return `
      <div style="display: flex; gap: 1rem; align-items: center;">
        <div class="skeleton skeleton-circle" style="width: 80px;
        height: 80px;"></div>
        <div style="flex: 1;">
          <div class="skeleton skeleton-line" style="width:
          200px;"></div>
          <div class="skeleton skeleton-line" style="width:
          150px;"></div>
        </div>
      </div>
    `;
  }
}

customElements.define('skeleton-loader', SkeletonLoader);
```

Usage:

```
<skeleton-loader type="card" lines="3"></skeleton-loader>
<skeleton-loader type="list" lines="5"></skeleton-loader>
<skeleton-loader type="profile"></skeleton-loader>
```

26.20 Recipe 20: Countdown Timer

Countdown timer with custom formatting.

```
class CountdownTimer extends HTMLElement {
  connectedCallback() {
    this.targetDate = new
        Date(this.getAttribute('target')).getTime();
    this.format = this.getAttribute('format') || 'dhms'; // days,
        hours, minutes, seconds
    this.render();
    this.startCountdown();
  }

  render() {
    this.innerHTML = `
      <style>
        .countdown {
          display: flex;
          gap: 1rem;
          font-family: monospace;
        }
        .time-unit {
          text-align: center;
        }
        .value {
          display: block;
          font-size: 2rem;
          font-weight: bold;
        }
        .label {
          display: block;
          font-size: 0.875rem;
          color: #666;
          text-transform: uppercase;
        }
        .expired {
```

```
          color: #f44336;
          font-size: 1.5rem;
        }
      </style>
      <div class="countdown"></div>
    `;

    this.container = this.querySelector('.countdown');
  }

  startCountdown() {
    this.updateDisplay();
    this.interval = setInterval(() => this.updateDisplay(), 1000);
  }

  updateDisplay() {
    const now = Date.now();
    const distance = this.targetDate - now;

    if (distance < 0) {
      this.container.innerHTML = '<div class="expired">Time
        expired!</div>';
      clearInterval(this.interval);
      this.pan.dispatch('countdown:expired', { target:
        this.targetDate });
      return;
    }

    const days = Math.floor(distance / (1000 * 60 * 60 * 24));
    const hours = Math.floor((distance % (1000 * 60 * 60 * 24)) /
        (1000 * 60 * 60));
    const minutes = Math.floor((distance % (1000 * 60 * 60)) / (1000
        * 60));
    const seconds = Math.floor((distance % (1000 * 60)) / 1000);
```

```
    const parts = [];
    if (this.format.includes('d')) parts.push({ value: days, label:
        'Days' });
    if (this.format.includes('h')) parts.push({ value: hours,
        label: 'Hours' });
    if (this.format.includes('m')) parts.push({ value: minutes,
        label: 'Minutes' });
    if (this.format.includes('s')) parts.push({ value: seconds,
        label: 'Seconds' });

    this.container.innerHTML = parts
      .map(p => `
        <div class="time-unit">
          <span class="value">${String(p.value).padStart(2,
        '0')}</span>
          <span class="label">${p.label}</span>
        </div>
      `)
      .join('');
  }

  disconnectedCallback() {
    if (this.interval) {
      clearInterval(this.interval);
    }
  }
}

customElements.define('countdown-timer', CountdownTimer);
```

Usage:

```
<countdown-timer target="2024-12-31T23:59:59"
        format="dhms"></countdown-timer>
```

26.21 Recipe 21: Progress Bar

Visual progress indicator with customization.

```
class ProgressBar extends HTMLElement {
  connectedCallback() {
    this.attachShadow({ mode: 'open' });
    this.value = parseFloat(this.getAttribute('value')) || 0;
    this.max = parseFloat(this.getAttribute('max')) || 100;
    this.showLabel = this.hasAttribute('show-label');
    this.render();

    this.pan.subscribe('progress:update', (event) => {
      if (event.detail.id === this.id) {
        this.updateProgress(event.detail.value);
      }
    });
  }

  render() {
    const percent = (this.value / this.max) * 100;

    this.shadowRoot.innerHTML = `
      <style>
        .progress-container {
          width: 100%;
          background: #e0e0e0;
          border-radius: 1rem;
          overflow: hidden;
          position: relative;
        }
        .progress-bar {
          height: 2rem;
          background: linear-gradient(90deg, #4caf50, #8bc34a);
          border-radius: 1rem;
          transition: width 0.3s ease;
```

```
          display: flex;
          align-items: center;
          justify-content: center;
          color: white;
          font-weight: bold;
        }
        .label {
          position: absolute;
          width: 100%;
          text-align: center;
          line-height: 2rem;
          color: ${percent > 50 ? 'white' : '#333'};
          z-index: 1;
        }
      </style>
      <div class="progress-container">
        ${this.showLabel ? `<div
        class="label">${percent.toFixed(0)}%</div>` : ''}
        <div class="progress-bar" style="width: ${percent}%">
        </div>
      </div>
    `;
  }

  updateProgress(value) {
    this.value = value;
    this.setAttribute('value', value);
    this.render();

    if (this.value >= this.max) {
      this.pan.dispatch('progress:complete', { id: this.id });
    }
  }
```

```
  static get observedAttributes() {
    return ['value'];
  }

  attributeChangedCallback(name, oldValue, newValue) {
    if (name === 'value' && oldValue !== newValue) {
      this.value = parseFloat(newValue);
      this.render();
    }
  }
}

customElements.define('progress-bar', ProgressBar);
```

Usage:

```
<progress-bar id="upload-progress" value="45" max="100" show-
        label></progress-bar>

<script>
// Update progress
pc.publish('progress:update', { id: 'upload-progress', value: 75 });
</script>
```

26.22 Common Patterns

26.22.1 Pattern: Component Composition

Build complex components from simpler ones.

```
class UserProfile extends HTMLElement {
  connectedCallback() {
    this.innerHTML = `
      <user-avatar user-id="${this.getAttribute('user-id')}"></user-
        avatar>
      <user-details user-id="${this.getAttribute('user-
        id')}"></user-details>
      <user-actions user-id="${this.getAttribute('user-
        id')}"></user-actions>
    `;
  }
}
```

26.22.2 Pattern: Higher-Order Components

Wrap components with additional functionality.

```
function withLoading(ComponentClass) {
  return class extends ComponentClass {
    connectedCallback() {
      this.showLoader();
      super.connectedCallback();
    }

    showLoader() {
      this.innerHTML = '<div class="loader">Loading...</div>';
    }
  };
}

customElements.define('user-card', withLoading(UserCard));
```

26.22.3 Pattern: Singleton Services

Share a single instance across components.

```
class DataCache {
  static instance = null;

  static getInstance() {
    if (!DataCache.instance) {
      DataCache.instance = new DataCache();
    }
    return DataCache.instance;
  }

  constructor() {
    this.cache = new Map();
  }

  get(key) {
    return this.cache.get(key);
  }

  set(key, value) {
    this.cache.set(key, value);
  }
}
```

26.23 Anti-Patterns to Avoid

26.23.1 Anti-Pattern: Tight Coupling

Bad:

```
class ComponentA extends HTMLElement {
  connectedCallback() {
    document.querySelector('component-b').doSomething();
  }
}
```

Good:

```
class ComponentA extends HTMLElement {
  connectedCallback() {
    this.pan.dispatch('action:requested', { data });
  }
}
```

26.23.2 Anti-Pattern: Massive Components

Bad: 500-line components handling everything.

Good: Break into focused, single-responsibility components.

26.23.3 Anti-Pattern: Ignoring Lifecycle

Bad:

```
class BadComponent extends HTMLElement {
  constructor() {
    super();
    this.innerHTML = '<div>Content</div>'; // Too early!
  }
}
```

Good:

```
class GoodComponent extends HTMLElement {
  connectedCallback() {
    this.innerHTML = '<div>Content</div>';
  }
}
```

26.23.4 Anti-Pattern: Manual Memory Leaks

Bad:

```
connectedCallback() {
  this.pan.subscribe('event', handler);
  // Never unsubscribed!
}
```

Good:

```
connectedCallback() {
  this.unsubscribe = this.pan.subscribe('event', handler);
}

disconnectedCallback() {
  this.unsubscribe();
}
```

These recipes provide battle-tested solutions for common scenarios. Adapt them to your needs, understanding the principles behind each pattern. The best code is readable, maintainable, and solves the problem at hand without unnecessary complexity.

27 Glossary

This glossary defines technical terms, LARC-specific concepts, and web standards references used throughout this manual. Terms are presented in alphabetical order with clear definitions and, where relevant, cross-references to related concepts.

27.1 A

Adapter A design pattern that converts one interface to another. In LARC contexts, adapters bridge LARC components with external libraries or non-standard APIs.

Attribute An HTML element property set via markup (e.g., `<my-component data-id="123">`). LARC components observe attributes through `observedAttributes` and respond to changes via `attributeChangedCallback()`.

Autonomous Custom Element A Web Component that extends `HTMLElement` directly rather than extending built-in HTML elements. LARC components are autonomous custom elements. Compare with *Customized Built-in Element*.

27.2 B

Batch Update An optimization technique that groups multiple state changes into a single render cycle, reducing unnecessary DOM operations and improving performance.

Binding The connection between a data source and its visual representation. LARC uses event-driven updates rather than automatic data binding, giving developers explicit control over rendering.

Browser Event Standard DOM events like `click`, `input`, or `submit`. LARC components listen to browser events and can translate them into PAN bus events for application-wide communication.

Bubble Event propagation through the DOM tree from child to parent elements. Browser events bubble by default; custom events must explicitly enable bubbling via `bubbles: true`.

27.3 C

Callback A function passed as an argument to another function, executed after a specific event or operation completes. LARC lifecycle methods (`connectedCallback`, `disconnectedCallback`) are callbacks invoked by the browser at specific times.

Composed An event property that determines whether the event crosses Shadow DOM boundaries. Set via `composed: true` in event initialization. Essential for events that need to traverse shadow roots.

Component A self-contained, reusable user interface element. In LARC, components are Web Components registered via `customElements.define()` and implementing standard lifecycle callbacks.

Custom Element The Web Components standard for creating new HTML elements with custom behavior. LARC applications are built from custom elements. See also *Autonomous Custom Element*.

Customized Built-in Element A Web Component that extends an existing HTML element (e.g., `<button is="fancy-button">`). LARC primarily uses autonomous custom elements rather than customized built-ins.

27.4 D

Declarative A programming style that describes *what* should happen rather than *how*. HTML templates are declarative. Contrast with *Imperative*.

Dependency Injection A pattern where dependencies are provided to a component rather than created internally. LARC components receive the PAN bus reference rather than accessing a global singleton.

Dispatch Sending an event to the PAN bus for other components to receive. Called via `this.pan.dispatch(eventType, detail)`.

DOM (Document Object Model) The browser's representation of an HTML document as a tree of objects. LARC components manipulate the DOM through standard APIs.

27.5 E

Element A node in the DOM tree representing an HTML tag. Custom elements are specialized elements with developer-defined behavior.

Emit Synonym for *dispatch*. Some frameworks use "emit" for event publication. LARC prefers "dispatch" to align with standard DOM terminology.

Encapsulation Hiding internal implementation details from external code. Shadow DOM provides style encapsulation; JavaScript class private fields provide data encapsulation.

Event A signal indicating something happened. Browser events (clicks, inputs) and custom events (PAN bus messages) both use the Event API.

Event Target Any object that can receive events and have listeners registered on it. All DOM nodes are event targets; the PAN bus is also an event target.

27.6 F

Fragment A `DocumentFragment` is a lightweight container for DOM nodes that can be manipulated off-screen and inserted into the document in one operation, reducing reflows.

Framework A comprehensive library providing structure and conventions for application development. LARC is a lightweight component architecture rather than a full framework, emphasizing web standards.

27.7 H

HTML Template The `<template>` element stores client-side content that won't render until explicitly instantiated. Useful for defining reusable markup structures.

Hydration The process of attaching event listeners and state to server-rendered HTML. LARC components hydrate automatically when defined via `customElements.define()`.

27.8 I

Imperative A programming style describing *how* to accomplish a task through explicit instructions. JavaScript is imperative. Contrast with *Declarative*.

Intersection Observer A browser API for efficiently detecting when elements enter or leave the viewport. Used for lazy loading, infinite scroll, and visibility tracking.

27.9 L

LARC (Lightweight Asynchronous Reactive Components) The component architecture described in this manual, emphasizing web standards, minimal abstraction, and explicit communication patterns.

Lifecycle The sequence of states a component passes through: creation, attachment to DOM, updates, and removal. LARC components implement standard Web Components lifecycle callbacks.

Lifecycle Callback Methods invoked by the browser at specific points in a component's lifecycle: `constructor()`, `connectedCallback()`, `disconnectedCallback()`, `attributeChangedCallback()`, `adoptedCallback()`.

Light DOM Regular DOM content, as opposed to Shadow DOM. Content placed inside a custom element's tags lives in the light DOM and can be redistributed via `<slot>`.

27.10 M

Microtask A JavaScript task scheduled via `Promise.then()` or `queueMicrotask()`. Microtasks run before the browser's next rendering cycle, useful for batching updates.

Module An ES6 module (`import` / `export`) that encapsulates code. LARC components are typically defined as modules, one component per file.

Mutation Observer A browser API for watching DOM changes. Less commonly needed in LARC since components manage their own rendering.

27.11 N

Namespace A prefix used to group related events or APIs. LARC encourages namespacing PAN bus events (e.g., `user:login`, `user:logout`) for better organization.

Node A basic DOM building block. Elements, text, and comments are all nodes. Components manipulate nodes through standard DOM APIs.

27.12 O

Observer Pattern A design pattern where objects (observers) subscribe to state changes in another object (subject). The PAN bus implements the observer pattern.

observedAttributes A static getter on custom element classes listing attributes the component wants to monitor. Changes trigger `attributeChangedCallback()`.

27.13 P

PAN Bus (Publish-and-subscribe Asynchronous Notification Bus) LARC's event system for component communication. Components dispatch events to the bus and subscribe to events they care about, enabling loose coupling.

Polyfill JavaScript code that implements modern features in older browsers. Web Components polyfills enable LARC applications to run in browsers without native support.

Prop (Property) Short for "property," data passed to a component. In LARC, complex data typically flows through PAN bus events rather than attributes, since attributes are limited to strings.

Publish-Subscribe A messaging pattern where publishers send messages to topics/channels, and subscribers receive messages from those topics. The PAN bus is a pub-sub system.

27.14 R

Reactive A programming model where the UI automatically updates in response to data changes. LARC components implement reactivity explicitly through PAN bus subscriptions rather than automatic binding.

Reconciliation The process of determining minimal DOM changes needed to reflect new state. LARC leaves reconciliation to developers or optional libraries rather than providing built-in virtual DOM diffing.

Render Convert data into visual representation. In LARC, rendering is explicit—components call their own `render()` methods when appropriate.

Reflow The browser's process of recalculating element positions and dimensions. Excessive reflows hurt performance. LARC's batch updates minimize reflows.

27.15 S

Scoped Styles CSS that applies only to a specific component without affecting other elements. Shadow DOM provides automatic style scoping.

Shadow DOM A web standard for attaching encapsulated DOM trees to elements. Shadow DOM provides style and markup encapsulation, preventing styles from leaking in or out.

Shadow Host The element to which a shadow root is attached. When you call `this.attachShadow()` on a custom element, that element becomes the shadow host.

Shadow Root The root of a shadow DOM tree, created via `element.attachShadow()`. Content inside the shadow root is isolated from the main document.

Slot A Shadow DOM feature for distributing light DOM content into shadow DOM. Defined with `<slot>` elements and allowing flexible content composition.

State Data that determines component appearance and behavior. LARC encourages explicit state management through component properties and PAN bus events.

Subscribe Registering a listener for events on the PAN bus. Called via `this.pan.subscribe(eventType, handler)`, returns an unsubscribe function.

27.16 T

Template Reusable markup structure. Can refer to HTML `<template>` elements or template literals (backtick strings) used for generating HTML.

Template Literal JavaScript's backtick string syntax supporting multiline strings and interpolation. Commonly used for component templates: `` `<div>${value}</div>` ``.

Throttle Limiting function execution frequency. Unlike debouncing (which delays until activity stops), throttling ensures a function runs at most once per time interval.

27.17 U

Unsubscribe Removing a listener from the PAN bus. The function returned by `subscribe()` acts as an unsubscribe callback, essential for preventing memory leaks.

User Agent The browser or other software accessing a web application. User agent strings identify the browser type and version.

27.18 V

Virtual DOM An in-memory representation of the DOM used to calculate minimal changes before applying them. LARC doesn't include built-in virtual DOM, preferring explicit control or optional libraries.

27.19 W

Web Component An umbrella term for three standards: Custom Elements, Shadow DOM, and HTML Templates. LARC applications are built on Web Components.

Web Standards Specifications maintained by standards bodies (W3C, WHATWG) defining how web technologies work. LARC prioritizes web standards over proprietary abstractions.

27.20 LARC-Specific Terms

Component Bus A dedicated PAN bus instance for a specific component subtree. Allows isolated event scopes within larger applications. Most applications use a single global bus.

Component Tree The hierarchical structure of custom elements in an application. Events and data flow through this tree via the PAN bus.

Event Detail The `detail` property of a custom event, containing application-specific data. PAN bus events place their payload in the detail object.

Event Type A string identifying an event category (e.g., `'user:login'`, `'data:loaded'`). LARC encourages namespaced, descriptive event types.

Pan Property The `pan` property on custom elements, providing access to the PAN

bus. Automatically injected by LARC's component initialization.

Reactive Primitive Basic reactive building blocks like reactive objects, computed values, and watchers. LARC's optional reactivity system provides these as lightweight utilities.

Unidirectional Data Flow An architecture where data flows in one direction through an application. LARC encourages this through PAN bus events: components dispatch actions upward and listen for state changes downward.

27.21 Web Standards References

CustomElementRegistry The browser's registry of defined custom elements, accessed via `window.customElements`. Provides `define()`, `get()`, `whenDefined()`, and `upgrade()` methods.

Event.prototype.composed Boolean property indicating whether an event crosses shadow DOM boundaries during event propagation.

Event.prototype.bubbles Boolean property indicating whether an event propagates up the DOM tree from its target.

HTMLElement The base interface for HTML elements. All LARC components extend `HTMLElement` or its subclasses.

MutationObserver API Interface for observing DOM mutations. Occasionally useful for LARC components that need to react to external DOM changes.

ShadowRoot Interface representing the root of a shadow DOM tree, providing methods like `querySelector()` that operate within the shadow scope.

shadowRoot.mode The encapsulation mode of a shadow root: `'open'` (accessible via `element.shadowRoot`) or `'closed'` (inaccessible from outside). LARC recommends open mode for testability.

27.22 Acronyms and Abbreviations

API - Application Programming Interface **CDN** - Content Delivery Network **CSS** - Cascading Style Sheets **DOM** - Document Object Model **ES6** - ECMAScript 2015 (JavaScript version) **HTML** - HyperText Markup Language **HTTP** - HyperText Transfer Protocol **JSX** - JavaScript XML (React's template syntax, not part of LARC) **LARC** - Lightweight Asynchronous Reactive Components **MVC** - Model-View-Controller **NPM** - Node Package Manager **PAN** - Publish-and-subscribe Asynchronous Notification **REST** - Representational State Transfer **SPA** - Single-Page Application **SSR** - Server-Side Rendering **UI** - User Interface **URL** - Uniform Resource Locator **VDOM** - Virtual DOM **W3C** - World Wide Web Consortium **WHATWG** - Web Hypertext Application Technology Working Group

27.23 Related Concepts

Component Lifecycle See *Lifecycle* and *Lifecycle Callback*.

Custom Events Events created via `new CustomEvent()` rather than browser-generated events. PAN bus events are custom events.

Event-Driven Architecture An architectural pattern where components communicate through events rather than direct method calls. LARC's PAN bus enables event-driven architecture.

Loose Coupling Design principle where components depend on abstractions (event types) rather than concrete implementations (specific components), making systems more flexible and maintainable.

Separation of Concerns Design principle where different aspects of functionality are handled by different components. LARC components encapsulate specific UI concerns, communicating via well-defined events.

Single Responsibility Principle Each component should have one clear purpose. LARC encourages focused components that do one thing well.

27.24 Further Reading

MDN Web Docs (developer.mozilla.org) Comprehensive reference for Web APIs, including Web Components, DOM manipulation, and JavaScript features.

Web Components Specifications Official standards documents at w3.org and whatwg.org defining Custom Elements, Shadow DOM, and HTML Templates.

LARC Documentation Complete API reference and guides at larc.dev.

ECMAScript Specifications JavaScript language specifications at tc39.es.

This glossary covers core concepts needed to work effectively with LARC. For deeper exploration of specific topics, consult the main chapters of this manual and the reference materials listed above. Understanding these terms and their relationships helps you write clearer code, communicate more effectively with other developers, and leverage the full power of web standards in your applications.

28 Resources

This appendix provides a curated collection of resources for learning, using, and extending LARC. Whether you're getting started, troubleshooting a problem, or contributing to the ecosystem, these links will help you find what you need.

28.1 Official Documentation

28.1.1 Primary Documentation

LARC Core Repository https://github.com/larcjs/larc The main LARC repository containing the core framework source code, examples, and technical documentation. This is the authoritative source for implementation details and includes the complete test suite.

LARC Components Library https://github.com/larcjs/larc/tree/main/packages/components Official component library with production-ready UI components, data components, integration components, and utilities. Each component includes comprehensive documentation and working examples.

API Reference https://larcjs.com/api Complete API documentation for all core classes, components, and utilities. Includes type definitions, method signatures, and interactive examples.

Getting Started Guide https://larcjs.com/getting-started Quick-start guide for new developers. Walks through installation, first application, and core concepts in 30 minutes.

28.1.2 Companion Books

Learning LARC The tutorial-focused companion to this reference manual. Organized around progressive learning with hands-on exercises, projects, and quizzes. Ideal for developers new to LARC or component-based architecture.

Building with LARC: A Reference Manual This book. Comprehensive reference covering all aspects of LARC development from core concepts to advanced patterns. Available online at https://larcjs.com/reference

28.2 Community Resources

28.2.1 Forums and Discussion

LARC Discussions (GitHub) https://github.com/larcjs/larc/discussions Official discussion forum for LARC developers. Ask questions, share projects, discuss patterns, and connect with other developers. Monitored by core maintainers.

Stack Overflow https://stackoverflow.com/questions/tagged/larc Tag: `larc` For technical troubleshooting and specific programming questions. Search existing questions before posting new ones.

Discord Community https://discord.gg/zjUPsWTu Real-time chat for LARC developers. Channels for beginners, advanced topics, component development, and off-topic discussion. Most active community hub.

Reddit r/larcjs https://reddit.com/r/larcjs Community-run subreddit for LARC news, showcases, and discussion. Good for project feedback and ecosystem updates.

28.2.2 Social Media

Twitter/X: @larcjs https://twitter.com/larcjs Official Twitter account for

announcements, tips, and community highlights. Follow for news about releases, events, and ecosystem updates.

Mastodon: @larcjs@fosstodon.org https://fosstodon.org/@larcjs Official presence on the Fediverse for developers who prefer open platforms.

LinkedIn: LARC Developers Group https://linkedin.com/groups/larcjs Professional network for LARC developers. Good for job postings, industry discussion, and enterprise use cases.

28.3 Code Examples and Templates

28.3.1 Example Applications

Official Examples Repository https://github.com/larcjs/larc/tree/main/packages/examples Curated collection of example applications demonstrating LARC patterns and components. Each example is self-contained, documented, and includes setup instructions.

Notable examples include:

- Task Manager (state management, persistence)
- E-commerce Store (routing, forms, authentication)
- Real-time Chat (WebSocket integration, presence)
- File Manager (OPFS, drag-and-drop, uploads)
- Dashboard Builder (composable widgets, theming)

CodeSandbox Templates https://codesandbox.io/search?refinementList%5Btags%5D=larc Interactive online templates for rapid prototyping. Fork and experiment without local setup. Includes starter templates for common application types.

GitHub Topics: #larcjs https://github.com/topics/larcjs Community-contributed

projects using LARC. Browse for inspiration, study real-world implementations, and discover reusable components.

28.3.2 Component Showcases

LARC Component Gallery https://components.larcjs.com Visual gallery of all official components with live demos, code samples, and customization tools. Essential reference when choosing components for your project.

Awesome LARC Components https://github.com/larcjs/awesome-larc-components Curated list of community-built components. Organized by category (UI, data, integration) with quality ratings and maintenance status.

28.4 Development Tools

28.4.1 Browser Extensions

LARC DevTools Chrome: https://chrome.google.com/webstore/detail/larc-devtools Firefox: https://addons.mozilla.org/firefox/addon/larc-devtools Browser extension providing visual PAN message inspection, component tree visualization, performance profiling, and state debugging.

Web Components DevTools General-purpose extension for debugging all Web Components, including LARC components. Useful for inspecting Shadow DOM and custom element lifecycles.

28.4.2 Editor Extensions

VS Code: LARC Extension https://marketplace.visualstudio.com/items?itemName=larcjs.larc-vscode Official VS Code extension providing:

- Component auto-completion
- PAN topic IntelliSense
- Snippet library for common patterns
- Integrated component browser
- Live component preview

JetBrains Plugin: LARC Support https://plugins.jetbrains.com/plugin/larcjs-support Support for WebStorm, IntelliJ IDEA, and other JetBrains IDEs. Provides code completion, navigation, and refactoring tools.

28.4.3 Command-Line Tools

LARC CLI https://github.com/larcjs/larc/tree/main/cli

```
$ npm install -g create-larc-app
```

Official command-line interface for:

- Project scaffolding (`larc create`)
- Component generation (`larc generate`)
- Development server with hot reload (`larc dev`)
- Production builds (`larc build`)
- Component registry integration (`larc add`)

create-larc-app https://github.com/larcjs/larc/tree/main/cli

```
$ npx create-larc-app my-app
```

Zero-configuration starter for new LARC projects. Includes pre-configured development environment, example components, and build tooling.

28.5 Learning Resources

28.5.1 Video Tutorials

LARC Fundamentals (YouTube) https://youtube.com/playlist?list=PLarc-fundamentals Official video series covering:

- Introduction to LARC (15 min)
- PAN Bus Messaging (22 min)
- Component Development (28 min)
- State Management Patterns (35 min)
- Building a Complete Application (1h 15min)

Egghead.io: Building with LARC https://egghead.io/courses/building-with-larc Professional screencast series (paid) with bite-sized lessons on specific topics. High production quality with accompanying code repositories.

Frontend Masters: LARC Workshop https://frontendmasters.com/courses/larc Full-day workshop covering LARC from fundamentals to advanced patterns. Includes exercises, quizzes, and downloadable resources.

28.5.2 Blog Posts and Articles

LARC Blog https://blog.larcjs.com Official blog with deep dives into architecture decisions, release notes, performance analysis, and best practices from core maintainers.

CSS-Tricks: LARC Guide Series https://css-tricks.com/guides/larc Multi-part guide covering LARC from a frontend developer's perspective. Excellent for understanding how LARC fits into modern web development.

Smashing Magazine: Component Architecture with LARC

https://smashingmagazine.com/larc-component-architecture In-depth article comparing LARC's approach to other component frameworks. Good for understanding trade-offs and architectural decisions.

28.5.3 Podcasts

Syntax.fm: LARC Deep Dive https://syntax.fm/show/larc-deep-dive Popular web development podcast featuring LARC's creator discussing philosophy, implementation, and future direction.

ShopTalk Show: Building without Build Tools https://shoptalkshow.com/larc-episode Discussion of zero-build development philosophy and LARC's approach to modern web development.

28.6 Related Projects and Technologies

28.6.1 Web Components Standards

Web Components at MDN https://developer.mozilla.org/en-US/docs/Web/Web_Components Comprehensive documentation for Custom Elements, Shadow DOM, HTML Templates, and related browser APIs that LARC builds upon.

webcomponents.org https://webcomponents.org Community hub for Web Components with tutorials, best practices, and a component directory. Not LARC-specific but highly relevant.

Custom Elements Everywhere https://custom-elements-everywhere.com Test suite showing how different frameworks work with Web Components. Demonstrates LARC's excellent interoperability.

28.6.2 Message Bus Patterns

Enterprise Integration Patterns: Messaging https://enterpriseintegrationpatterns.com/patterns/messaging Classic reference for message-based architecture patterns. LARC implements many patterns described here adapted for browser environments.

Reactive Manifesto https://reactivemanifesto.org Principles of reactive system design that influenced LARC's architecture, particularly around message-driven communication.

28.6.3 Complementary Technologies

IndexedDB API https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API Browser database API used by LARC storage components for client-side persistence.

Origin Private File System (OPFS) https://developer.mozilla.org/en-US/docs/Web/API/File_System_Access_API Modern file system API supported by LARC file management components.

BroadcastChannel API https://developer.mozilla.org/en-US/docs/Web/API/Broadcast_Channel_API Cross-tab communication API used by LARC for multi-window synchronization.

28.7 Backend Integration

28.7.1 LARC-Compatible Backends

Node.js Backend Examples https://github.com/larcjs/larc/tree/main/packages/examples/backends/nodejs Reference implementations showing REST and WebSocket backends for LARC applications. Includes authentication, file uploads, and real-time

features.

Python/Flask Backend Examples https://github.com/larcjs/larc/tree/main/packages/examples/backends/python Python backend examples demonstrating API design patterns that work well with LARC frontend applications.

Deno Backend Examples https://github.com/larcjs/larc/tree/main/packages/examples/backends/deno Modern JavaScript runtime examples showing how to build backends without Node.js dependencies.

28.7.2 API Design Guides

REST API Design for LARC Applications https://larcjs.com/guides/rest-api-design Best practices for designing REST APIs that integrate cleanly with LARC's data components and message patterns.

WebSocket Integration Guide https://larcjs.com/guides/websocket-integration How to implement real-time features using WebSocket connections with LARC's messaging system.

28.8 Testing and Quality

28.8.1 Testing Resources

LARC Testing Guide https://larcjs.com/guides/testing Official guide for testing LARC applications covering unit tests, integration tests, end-to-end tests, and visual regression testing.

Web Test Runner https://modern-web.dev/docs/test-runner/overview Recommended test runner for LARC applications. Fast, supports Web Components natively, and requires no browser driver.

Playwright https://playwright.dev End-to-end testing framework recommended for LARC application testing. Excellent Web Component support and debugging tools.

28.8.2 Performance Resources

Web Performance Working Group https://w3c.github.io/web-performance W3C standards for measuring and optimizing web performance. LARC follows these standards for component performance metrics.

web.dev Performance https://web.dev/performance Google's comprehensive performance guide covering Core Web Vitals, optimization techniques, and measurement tools relevant to LARC applications.

28.9 Contributing and Extending

28.9.1 Contribution Guides

Contributing to LARC Core https://github.com/larcjs/larc/blob/main/CONTRIBUTING.md Guidelines for contributing to the LARC core framework. Includes coding standards, testing requirements, and pull request process.

Publishing Components https://larcjs.com/guides/publishing-components How to create, document, and publish reusable LARC components for the community. Covers naming conventions, versioning, and registry submission.

Component Development Guide https://larcjs.com/guides/component-development Best practices for building high-quality LARC components including accessibility, performance, and API design.

28.9.2 Governance and Roadmap

LARC Roadmap https://github.com/larcjs/larc/blob/main/ROADMAP.md Public roadmap showing planned features, architectural improvements, and long-term vision. Community feedback welcome.

RFC Process https://github.com/larcjs/rfcs Request for Comments process for proposing major changes to LARC. Review active RFCs and submit your own proposals.

Governance Model https://github.com/larcjs/larc/blob/main/GOVERNANCE.md How LARC is governed, who makes decisions, and how the community can participate in the project's direction.

28.10 Package Registries

28.10.1 NPM Packages

@larcjs/core https://npmjs.com/package/@larcjs/core Core framework package containing PAN bus, autoloader, and foundational components.

@larcjs/ui https://npmjs.com/package/@larcjs/ui Official component library with UI, data, and integration components.

@larcjs/core-types https://npmjs.com/package/@larcjs/core-types TypeScript type definitions for LARC core APIs and components.

@larcjs/testing-library https://npmjs.com/package/@larcjs/testing-library Testing utilities and helpers for LARC applications.

28.10.2 CDN Distributions

unpkg.com https://unpkg.com/@larcjs/core@latest Fast, global CDN for quick prototyping and development. Automatically serves latest versions.

jsDelivr https://cdn.jsdelivr.net/npm/@larcjs/core@latest Alternative CDN with excellent performance and reliability. Supports version pinning and package exploration.

LARC Official CDN https://cdn.larcjs.com Official CDN optimized for LARC with guaranteed uptime, geographic distribution, and versioned URLs.

28.11 Books and Long-Form Resources

28.11.1 Recommended Reading

Web Components: From Zero to Hero By Pascal Schilp Foundation knowledge for understanding the Web Components standards that LARC builds upon. Available free online.

Component-Based Development in JavaScript By Oliver Steele Explores component architecture patterns with examples in multiple frameworks including LARC. Good for understanding architectural trade-offs.

Event-Driven Architecture By Martin Fowler Classic software architecture text covering message-based patterns that inform LARC's design philosophy.

28.11.2 Academic Papers

Web Components: Standards, Patterns, and Best Practices Research paper analyzing Web Components adoption and patterns. Includes LARC case studies.

Message-Oriented Middleware for Browser Applications Academic treatment of

message bus patterns in web applications with LARC as example implementation.

28.12 Deployment and Hosting

28.12.1 Hosting Platforms

Netlify https://netlify.com Recommended static hosting platform for LARC applications. Free tier suitable for most projects. Excellent CDN and deployment pipeline.

Vercel https://vercel.com Alternative hosting platform with Git integration, preview deployments, and serverless functions for backend features.

Cloudflare Pages https://pages.cloudflare.com Global edge network hosting with fast deployments and excellent performance. Good for international applications.

GitHub Pages https://pages.github.com Free hosting for open source projects. Simple deployment directly from GitHub repositories.

28.12.2 Deployment Guides

LARC Deployment Guide https://larcjs.com/guides/deployment Comprehensive guide covering deployment options, optimization strategies, caching configuration, and production best practices.

Performance Optimization Guide https://larcjs.com/guides/performance-optimization How to optimize LARC applications for production including code splitting, lazy loading, asset optimization, and CDN configuration.

28.13 Events and Training

28.13.1 Conferences

LARC Conf https://conf.larcjs.com Annual conference dedicated to LARC featuring talks, workshops, and networking. Recordings available online.

Web Components Summit https://webcomponentssummit.com General Web Components conference with LARC-specific tracks and presentations.

28.13.2 Workshops and Training

Official LARC Workshops https://larcjs.com/workshops In-person and virtual workshops taught by LARC experts. Topics range from fundamentals to advanced patterns.

Corporate Training https://larcjs.com/training Customized training programs for enterprise teams. Includes on-site workshops, consultation, and ongoing support.

28.14 Community Projects

28.14.1 Notable Applications Built with LARC

Browse https://larcjs.com/showcase for featured applications demonstrating LARC's capabilities in production environments.

28.14.2 Open Source Projects

LARC DevTools Browser extension and debugging toolkit (open source)

LARC Component Library Templates Starter templates for building your own component libraries

LARC Form Builder Visual form builder with code generation

LARC Dashboard Framework Composable dashboard system with widgets and layouts

28.15 Stay Updated

28.15.1 Newsletters

LARC Weekly https://larcjs.com/newsletter Weekly newsletter covering LARC news, tutorials, community projects, and ecosystem updates.

Web Components Weekly https://webcomponents.dev/newsletter General Web Components newsletter that frequently features LARC content.

28.15.2 Release Notes

LARC Changelog https://github.com/larcjs/larc/blob/main/CHANGELOG.md Detailed changelog for all LARC releases including breaking changes, new features, and bug fixes.

Security Advisories https://github.com/larcjs/larc/security/advisories Security announcements and vulnerability reports. Subscribe for critical updates.

28.16 Getting Help

When you need assistance:

1. **Search existing resources**: Check documentation, Stack Overflow, and GitHub Discussions first
2. **Prepare a minimal reproduction**: Create a CodeSandbox or GitHub repo demonstrating your issue
3. **Be specific**: Include LARC version, browser, error messages, and what you've already tried
4. **Choose the right channel**:
    - Technical questions -> Stack Overflow (tag: `larc`)
    - Bug reports -> GitHub Issues
    - General discussion -> GitHub Discussions or Discord
    - Real-time help -> Discord #help channel

28.16.1 Support Options

Community Support (Free) Discord, GitHub Discussions, Stack Overflow

Professional Support https://larcjs.com/support Commercial support plans available for enterprise users needing guaranteed response times and consulting.

This appendix is maintained by the LARC community. To suggest additions or corrections, submit a pull request to https://github.com/larcjs/larc-docs or open an issue describing the change.

Last updated: December 2025

29 Index

29.1 A

29.2 B

29.3 C

29.4 D

29.5 E

29.6 F

29.7 G

29.8 H

29.9 I

29.10 J

29.11 K

29.12 L

29.13 M

29.14 N

Naming conventions

29.15 O

29.16 P

29.17 Q

29.18 R

29.19 S

29.20 T

29.21 U

29.22 V

29.23 W

29.24 X

29.25 Y

29.26 Z

29.27 Appendices

29.28 Components Quick Reference

- pan-bus
- pan-theme-provider
- pan-theme-toggle
- pan-routes

Data Components (Chapter 18)

- pan-store
- pan-storage
- pan-data-table
- pan-list
- pan-filter
- pan-sort

UI Components (Chapter 19)

- pan-button
- pan-card
- pan-modal
- pan-toast
- pan-tabs
- pan-accordion
- pan-markdown-editor

Integration Components (Chapter 20)

- pan-http
- pan-websocket
- pan-sse
- pan-auth
- pan-analytics

Utility Components (Chapter 21)

- pan-logger
- pan-validator

This index references chapters and appendices by title. Page numbers would be added in print editions.

30 Colophon

30.1 About the Cover Animal

The animal on the cover of *Building with LARC* is a **North American Beaver** (*Castor canadensis*), one of nature's most accomplished engineers and builders. Beavers are large, semi-aquatic rodents native to North America, known for their remarkable ability to modify their environment through the construction of dams, lodges, and canals.

Adult beavers typically weigh between 35-65 pounds and measure 3-4 feet in length, including their distinctive flat, paddle-shaped tail. Their dense, waterproof fur provides excellent insulation in cold water, while webbed hind feet make them powerful swimmers. The beaver's most recognizable features are its large, continuously growing incisors - sharp orange teeth that can fell trees up to 3 feet in diameter.

Beavers are legendary for their construction skills. Using branches, logs, mud, and stones, they build elaborate dams that create ponds and wetlands, fundamentally reshaping their ecosystem. These structures can span hundreds of feet and last for decades, sometimes even centuries with regular maintenance. The engineering is sophisticated: dams are typically built with a curved shape to better withstand water pressure, and beavers continuously monitor and repair their structures, plugging leaks and reinforcing weak points.

Their lodges - dome-shaped homes built from sticks and mud - feature underwater entrances for protection from predators, with living chambers positioned above the waterline. Inside, these chambers remain remarkably warm even during harsh winters, thanks to excellent insulation and the occupants' body heat.

Beavers work primarily at night and are highly industrious, with family groups collaborating on construction projects. They communicate through tail slaps on the

water (warning signals), scent marking, and vocalizations. Their tree-felling technique is methodical: they gnaw around the trunk in an hourglass pattern until the tree falls, then section it into manageable pieces for transport and use.

We chose the Beaver for this book because, like these master builders, software architecture requires careful planning, solid engineering principles, and the ability to construct complex systems from simple components. The beaver's methodical approach to building - starting with a foundation, reinforcing structures, and creating systems that serve multiple purposes - mirrors the best practices in software development. Just as beaver dams create thriving ecosystems, well-built applications create value for entire communities of users.

30.2 About the Cover Illustration

The cover illustration was hand-drawn by the author using pen and ink, photographed, and refined in Photoshop. The woodcut-style engraving technique echoes the natural history illustrations of 18th and 19th century field guides and the iconic animal covers of O'Reilly Media's technical books. The beaver is depicted sitting alertly on a log with wood chips nearby - a working builder surveying its domain - which seemed particularly fitting for a reference manual about building applications. The detailed cross-hatching captures the texture of the beaver's fur and the grain of the wood, reflecting both the precision of the craft and the artistry involved.

30.3 Fonts

The text typeface is Adobe Minion Pro, the display typeface is Myriad Pro, and the code typeface is Ubuntu Mono. The book was typeset using Pandoc and LaTeX.

30.4 Production Notes

This book was authored in Markdown using a modular structure with 25 chapters, 7 appendices, and a comprehensive index. The content was converted to multiple formats (HTML, PDF, and EPUB) using Pandoc. All code examples were tested against LARC version 3.x and validated for correctness.

The book follows O'Reilly's tradition of comprehensive technical documentation: starting with philosophy and context, moving through practical implementation, and concluding with exhaustive API reference material. The structure was inspired by classic programming references like *Programming Perl* and *Programming Python*, which balance narrative explanation with detailed technical specifications.

The build system itself was designed to embody LARC's philosophy: it works without complex toolchains, uses standard tools (Pandoc, LaTeX), and produces professional results with minimal configuration. In a meta sense, the build script is a small LARC application - taking structured inputs and producing multiple coordinated outputs.

30.5 Acknowledgments

Special thanks to the LARC community for their contributions, feedback, and real-world usage patterns that informed many sections of this book. Thanks also to Christopher Robison for his vision in creating LARC and his commitment to web standards and developer ergonomics.

If you find an error in this book, have suggestions for improvements, or want to contribute additional recipes and patterns, please visit our GitHub repository or contact the author. This book is a living document that grows with the LARC community.

31 Learning LARC

31.1 The Web Has Grown Up. It’s Time Our Apps Did Too.

Modern browsers aren’t the brittle playgrounds they once were. They’re fast, secure, richly capable application platforms — yet most of today’s development stacks still treat them like dumb terminals that need layers of tooling, bundling, and framework magic just to function.

Learning LARC shows another path.

LARC embraces the browser as a mature runtime, using nothing but open standards — Web Components, modules, events, and message buses — to build complex, deeply interactive applications without build systems, without monoliths, and without ceremony. Through clear narrative examples and real architectural stories, this book teaches you how to design apps as ecosystems: small parts, clearly defined, communicating through a shared bus.

You’ll learn how to structure large systems out of tiny cooperating modules, expose capabilities through message patterns instead of global state, keep your interfaces clean, and let the platform do the heavy lifting it was built for.

No bundlers. No scaffolding. No twenty-layer dependency stacks. Just the browser, finally treated like the grown-up it is.

Whether you’re maintaining a legacy system or starting fresh, **Learning LARC** will help you rethink how modern web apps can — and should — be built.

31.2 About the Author

Christopher Robison is a veteran software engineer and architect with nearly three decades of experience building systems that range from biotech and online trading platforms to complex web applications and AI-driven tools. A lifelong maker with a deep appreciation for open standards, he has spent his career exploring the boundaries of what the web can do when you stop fighting the platform and start embracing it.

He is the creator of LARC.js and the PAN message bus, a browser-native architecture inspired by the elegant simplicity of the automotive CAN bus. His work blends engineering pragmatism with a playful curiosity that has led him to design everything from 3D printers and robotics to interactive music systems and decentralized applications.

Christopher currently lives in San Francisco, where he continues to build things that bridge the digital and physical worlds — and occasionally sneaks off to play punk rock shows with his band.

Website: https://larcjs.com (https://larcjs.com)