Below is a **clean, simplified, internally accurate, and NoteLM-friendly version** of **Custom Events in JavaScript**, written in the **same depth + structure** as your previous Event notes (Bubbling, Capturing, Delegation).
I've **removed unnecessary fluff**, clarified **internal working**, and kept **examples crisp and exam-ready**.

🎯 Custom Events in JavaScript

(Concept · Internal Working · Flow · Examples)

1️⃣ What are Custom Events?

Custom Events are **user-defined events** in JavaScript that allow different parts of an application to **communicate without direct coupling**.

They let one part of the code **signal that something has happened**, while other parts **listen and react** to that signal.

📌 Think of custom events as:

> "I don't know who is listening, but something important just happened."

2️⃣ Why Custom Events Exist (Internal Reason 🧠)

JavaScript's built-in events (click, load, keydown, etc.) are **limited to browser actions**.

Custom events exist to:

- Notify **application-level changes**
- Enable **loose coupling** between components
- Implement **Publish–Subscribe (Pub/Sub)** pattern
- Improve **scalability and maintainability**

3️⃣ Core Components of Custom Events

Component	Purpose
`Event` / `CustomEvent`	Creates a custom event
`addEventListener()`	Listens for the custom event
`dispatchEvent()`	Triggers (fires) the event
`event.type`	Name of the event
`event.detail`	Extra data (CustomEvent only)

4️⃣ Event vs CustomEvent (IMPORTANT)

Feature	Event	CustomEvent
Custom name	✅	✅
Pass data	❌	✅ (`detail`)
Extends	Base event	Extends `Event`
Use case	Simple signals	Data-based communication

📌 **Use `CustomEvent` when data must travel with the event**

5️⃣ Internal Flow of a Custom Event 🔁

```vbnet
1. Event is CREATED
2. Listener is REGISTERED
3. Event is DISPATCHED
4. Listener EXECUTES
```

✔️ Same propagation model

✔️ Supports capturing & bubbling

✔️ Behaves like native DOM events

6️⃣ Creating a Basic Custom Event

✅ Step 1: Create the Event

```js
const myEvent = new Event("myCustomEvent");
```

✅ Step 2: Listen for the Event

```js
document.addEventListener("myCustomEvent", () => {
  console.log("Custom event received!");
});
```

✅ Step 3: Dispatch the Event

```js
```

```
document.dispatchEvent(myEvent);
```

7️⃣ Basic Custom Event Example

```html
<button id="btn">Trigger Event</button>

<script>
const event = new Event("myCustomEvent");

document.addEventListener("myCustomEvent", () => {
  alert("Custom event triggered!");
});

document.getElementById("btn").addEventListener("click", () => {
  document.dispatchEvent(event);
});
</script>
```

✔️ Button click triggers **custom logic**

✔️ No direct function call

✔️ Loose coupling achieved

8️⃣ Custom Event with Data (`detail`) 📦

🧠 Why `detail`?

To send **extra information** with the event.

✅ Creating a CustomEvent with Data

```js
const customEvent = new CustomEvent("userLoggedIn", {
  detail: {
    username: "Anoop",
    role: "Student"
  }
});
```

✅ Listening & Accessing Data

```js
```

```
document.addEventListener("userLoggedIn", (event) => {
 console.log(event.detail.username);
 console.log(event.detail.role);
});
```

✅ Dispatching

```js
document.dispatchEvent(customEvent);
```

9️⃣ Example: Custom Event with Data (Complete)

```html
<button id="loginBtn">Login</button>

<script>
const loginEvent = new CustomEvent("loginSuccess", {
 detail: {
   message: "Login successful",
   time: new Date().toLocaleTimeString()
 }
});

document.addEventListener("loginSuccess", (event) => {
 alert(event.detail.message + " at " + event.detail.time);
});

document.getElementById("loginBtn").addEventListener("click", () => {
 document.dispatchEvent(loginEvent);
});
</script>
```

🔟 Condition-Based Custom Event Dispatching

📌 Custom events can be dispatched **conditionally**, just like normal logic.

✅ Example

```html
<script>
let role = "student";
```

```
const studentEvent = new Event("StudentEvent");
const adminEvent = new Event("AdminEvent");

document.addEventListener("StudentEvent", () => {
 alert("Welcome Student!");
});

document.addEventListener("AdminEvent", () => {
 alert("Welcome Admin!");
});

if (role === "student") {
 document.dispatchEvent(studentEvent);
} else {
 document.dispatchEvent(adminEvent);
}
</script>
```

✔️ Event depends on **runtime logic**

✔️ Highly flexible architecture

1️⃣1️⃣ Propagation in Custom Events

Custom events:

- ✔️ Bubble by default
- ✔️ Can be captured
- ✔️ Support `stopPropagation()`

```js
new CustomEvent("eventName", {
 bubbles: true,
 cancelable: true
});
```

1️⃣2️⃣ Custom Events & Pub/Sub Pattern 📡

Publisher

```js
document.dispatchEvent(new CustomEvent("dataReady"));
```

Subscriber

```js
document.addEventListener("dataReady", handler);
```

✔️ Publisher doesn't know subscribers

✔️ Subscribers don't know publisher

✔️ Clean separation of concerns

1️⃣3️⃣ Real-World Use Cases 🚀

✔️ Component communication

✔️ State updates

✔️ Login / logout notifications

✔️ Theme change signals

✔️ Framework internals

✔️ Game events

✔️ Analytics triggers

1️⃣4️⃣ Common Mistakes ❌

- Using `Event` when data is required
- Forgetting `detail` object
- Dispatching before listener is registered
- Using function calls instead of events

1️⃣5️⃣ Exam / Viva Ready Answers 🎓

Q: What is a custom event?

👉 A user-defined event used for application-level communication.

Q: Difference between Event and CustomEvent?

👉 CustomEvent allows data transfer using `detail`.

Q: How is a custom event triggered?

👉 Using `dispatchEvent()`.

1️⃣6️⃣ Key Takeaways ✅

✔️ Custom events enable loose coupling

✔️ `CustomEvent` supports data

✔️ Same propagation model as DOM events

✔️ Ideal for large applications

✔️ Cleaner than direct function calls

⏭️ What’s Next?

Type `next` to continue with:

👉 **stopPropagation vs stopImmediatePropagation**

👉 **Event Flow Diagram (Capture → Target → Bubble)**

👉 **Advanced Event Patterns (Once, Passive, Options)**