ECO 컴포넌트 런타임 가이드

컴포넌트가 동작하기 위해 런타임에서 주입받는 속성과 페이지 수준 인프라를 설명합니다.

> ***관련 문서***
>
> - *MIXIN_TUTORIAL.md — PopupMixin/HeatmapMixin step-by-step 튜토리얼*
> - *WKIT_API.md — bind3DEvents, fetchData, makeIterator 레퍼런스*
> - *EVENT_HANDLING.md — 이벤트 처리 원칙 (customEvents vs _internalHandlers)*
> - *INSTANCE_LIFECYCLE_GC.md — 인스턴스 생명주기 및 메모리 관리*

목차

---

1. 런타임 주입 속성 요약

register.js가 실행되는 시점에 `this`에 이미 주입되어 있는 속성입니다.

속성	타입	설명	사용 컴포넌트
`this.id`	string	컴포넌트 인스턴스 고유 ID	전체
`this.name`	string	인스턴스 이름 (빌더에서 설정)	ActionPanel
`this.setter`	object	빌더에서 설정한 속성 객체	3D 장비 컴포넌트
`this.properties`	object	퍼블리시 코드 등 정적 속성	3D 장비 컴포넌트
`this.page`	object	부모 페이지 인스턴스	전체
`this.appendElement`	Object3D	Three.js 3D 오브젝트	전체

전역 객체	속성	설명
`wemb.configManager`	`assetApiUrl`	API 서버 주소
`wemb.threeElements`	`camera`, `renderer`, `mainControls`	Three.js 렌더링 파이프라인

---

2. 속성별 상세

this.setter

빌더(에디터)에서 컴포넌트에 설정한 속성. 주로 자산 정보를 전달합니다.

```
// 모든 3D 장비 컴포넌트의 공통 패턴
this._defaultAssetKey = this.setter?.assetInfo?.assetKey || this.id;
```

경로	타입	설명
`setter.assetInfo`	object	자산 정보 객체
`setter.assetInfo.assetKey`	string	자산 식별 키 (API 호출에 사용)

ActionPanel은 `this.setter` 를 사용하지 않습니다. 대신 3D 레이어의 다른 컴포넌트를 순회하여 각 인스턴스의 assetKey를 가져옵니다.

---

this.properties

빌더에서 퍼블리시한 정적 데이터. 팝업 HTML/CSS 템플릿이 여기에 들어 있습니다.

```
const { htmlCode, cssCode } = this.properties.publishCode || {};
```

경로	타입	설명
properties.publishCode	object	Figma_Conversion 산출물
properties.publishCode.htmlCode	string	<template> 태그를 포함한 HTML
properties.publishCode.cssCode	string	Shadow DOM에 주입할 CSS

htmlCode 내부 구조:

```
<!-- component.html (Figma_Conversion 산출물) -->
<template id="popup-pdu">
  <div class="popup-container">
    <div class="popup-header">
      <span class="pdu-name"></span>
      <button class="close-btn">×</button>
    </div>
    <div class="chart-container"></div>
    <!-- ... -->
  </div>
</template>
```

register.js에서 `extractTemplate(htmlCode, 'popup-pdu')` 로 해당 template의 innerHTML을 추출합니다. template id는 컴포넌트별로 다릅니다:

컴포넌트	Template ID
PDU	popup-pdu
UPS	popup-ups
CRAC	popup-crac
SWBD	popup-swbd
TempHumiditySensor	popup-temp-humidity

---

this.page

부모 페이지 인스턴스. 데이터 서비스 접근과 컴포넌트 순회에 사용됩니다.

```
// 데이터 패치
fetchData(this.page, 'assetDetailUnified', { baseUrl, assetKey, locale });

// 3D 레이어 컴포넌트 순회 (ActionPanel 전용)
const iter = makeIterator(this.page, 'threeLayer');
for (const inst of iter) { ... }
```

용도	함수	설명
데이터 패치	fetchData(page, datasetName, param)	페이지에 등록된 데이터셋 호출

컴포넌트 순회	makeIterator(page, 'threeLayer')	3D 레이어의 모든 인스턴스 이터레이터

---

wemb.configManager

글로벌 설정 객체. 모든 컴포넌트에서 API 서버 주소를 가져올 때 사용합니다.

```
this._baseUrl = wemb.configManager.assetApiUrl.replace(/^https?:W/W//, '');
// 예: "https://api.example.com:4004" → "api.example.com:4004"
```

프로토콜(`https://`)을 제거하는 이유는 `fetchData` 내부에서 프로토콜을 별도로 처리하기 때문입니다.

---

wemb.threeElements

Three.js 렌더링 파이프라인 참조. 카메라 애니메이션이나 화면 좌표 변환에 사용됩니다.

```
const camera = wemb.threeElements.camera;
const controls = wemb.threeElements.mainControls;
const renderer = wemb.threeElements.renderer;
```

속성	타입	사용처
camera	THREE.Camera	카메라 위치/회전 애니메이션
mainControls	OrbitControls	카메라 타겟 제어
renderer	WebGLRenderer	3D→화면 좌표 변환

PDU에서 `zoomToFitGroup()` (장비 클릭 시 카메라 이동)과 `updateConnectionLine()` (3D 연결선 위치 계산)에서 사용합니다. 다른 컴포넌트는 직접 사용하지 않습니다.

---

3. 컴포넌트 유형별 의존성

유형 A: 기본 팝업 컴포넌트 (PDU, SWBD)

```
런타임 주입                          Mixin
─────────────────────────          ──────────
this.setter.assetInfo.assetKey     applyShadowPopupMixin
this.properties.publishCode        applyEChartsMixin
this.page
wemb.configManager.assetApiUrl
```

데이터셋:

데이터셋	갱신 주기	렌더 함수
assetDetailUnified	0 (1회)	renderBasicInfo
metricHistoryStats	5초	renderTrendChart

SWBD 특수 기능: 전력 탭에서 오늘/어제 비교 차트 (Promise.all 병렬 fetch)

유형 B: 팝업 + 상태카드 컴포넌트 (UPS)

```
런타임 주입                          Mixin
─────────────────────────          ──────────
this.setter.assetInfo.assetKey     applyShadowPopupMixin
this.properties.publishCode        applyEChartsMixin
```

```
this.page
wemb.configManager.assetApiUrl
```

데이터셋:

데이터셋	갱신 주기	렌더 함수
assetDetailUnified	0 (1회)	renderBasicInfo
metricLatest	5초	renderPowerStatus
metricHistoryStats	5초	renderTrendChart

유형 A와의 차이: `metricLatest` 데이터셋 추가, 상태카드(배터리 SOC, 잔여시간, 부하율, 배터리전압) 렌더링

유형 C: 팝업 + 히트맵 컴포넌트 (CRAC, TempHumiditySensor)

```
런타임 주입                          Mixin
─────────────────────────          ──────────
this.setter.assetInfo.assetKey     applyShadowPopupMixin
this.properties.publishCode        applyEChartsMixin
this.page                          applyHeatmapMixin
wemb.configManager.assetApiUrl
```

데이터셋:

데이터셋	갱신 주기	렌더 함수
assetDetailUnified	0 (1회)	renderBasicInfo
metricLatest	5초	renderStatusCards, renderIndicators
metricHistoryStats	5초	renderTrendChart

팝업 내 히트맵 토글 버튼:

```
// popupCreatedConfig.events
'.heatmap-btn': () => this.toggleHeatmap()
```

CRAC 추가 기능: 6개 BOOL 인디케이터 (팬, 냉방, 난방, 가습, 제습, 누수)

유형 D: 대시보드 패널 (ActionPanel)

ActionPanel은 PopupMixin을 사용하지 않는 유일한 컴포넌트입니다. 다른 3D 컴포넌트를 제어하는 "컨트롤러" 역할입니다.

```
런타임 주입                          Mixin
─────────────────────────          ──────────
this.page                          (없음 - HeatmapMixin은 centerInstance에 적용)
this.appendElement
```

사용하지 않는 속성:

속성	이유
`this.setter`	특정 자산에 종속되지 않음
`this.properties.publishCode`	팝업이 없음 (DOM 버튼만 사용)
`wemb.configManager`	자체 API 호출 없음 (순회 대상 인스턴스의 _baseUrl 사용)

핵심 런타임 속성:

```
// 히트맵을 적용할 대상 컴포넌트의 이름
this._centerComponentName = '';  // ← 런타임에서 외부 코드가 설정해야 함

// 3D 레이어 순회로 centerInstance 탐색
const iter = makeIterator(this.page, 'threeLayer');
for (const inst of iter) {
  if (inst.name === this._centerComponentName) {
    this._centerInstance = inst;
    break;
  }
}

// centerInstance에 HeatmapMixin 적용 (ActionPanel 자신이 아님!)
applyHeatmapMixin(this._centerInstance, {
  refreshInterval: 0,
  onLoadingChange: (isLoading) => { ... },
  ...HEATMAP_PRESET
});
```

ActionPanel의 동작 흐름:

```
ActionPanel 버튼 클릭 (온도분포도/온습도현황)
  |
  ├── activateMode('temperature') 또는 activateMode('humidity')
  |
  ├── [temperature 모드]
  |    ├── findCenterInstance()
  |    |    └── makeIterator(this.page, 'threeLayer')로 순회
  |    |         └── inst.name === this._centerComponentName 매칭
  |    ├── applyHeatmapMixin(centerInstance, HEATMAP_PRESET)
  |    └── centerInstance.toggleHeatmap()
  |
  ├── [humidity 모드]
  |    ├── makeIterator(this.page, 'threeLayer')로 전체 순회
  |    ├── 각 인스턴스의 _defaultAssetKey로 metricLatest fetch
  |    └── THREE.CSS2DObject로 3D 데이터 라벨 생성/갱신
  |
  └── ensureDataTimer()
       └── 30초 간격 통합 타이머 시작
            └── refreshAllActiveData()
                 ├── temperature: centerInstance.updateHeatmapWithData(points)
                 └── humidity: 라벨 텍스트 업데이트
```

---

4. ActionPanel 특수 패턴

_centerComponentName 설정

ActionPanel이 히트맵을 표시하려면 **어떤 3D 컴포넌트 위에 히트맵 서피스를 생성할지** 알아야 합니다. 이를 `_centerComponentName`으로 지정합니다.

```
// 런타임에서 외부 코드(또는 페이지 스크립트)가 설정
const actionPanel = /* 페이지에서 ActionPanel 인스턴스 참조 */;
actionPanel._centerComponentName = 'CRAC_CENTER';  // 3D 컴포넌트 name
```

이 값이 설정되지 않으면 히트맵 활성화 시 centerInstance를 찾지 못해 동작하지 않습니다.

통합 데이터 타이머

ActionPanel은 자체 `_refreshInterval` (기본 30초)로 **온도/습도 데이터를 한 번에 가져와** 히트맵과 라벨 양쪽에 분배합니다.

```
30초 타이머 (refreshAllActiveData)
  |
  ├── temperature 활성 시
  |     └── fetchData(page, 'metricLatest', { assetKey: centerInstance._defaultAssetKey })
  |           └── centerInstance.updateHeatmapWithData(dataPoints)
  |
  └── humidity 활성 시
        └── 3D 레이어 순회 → 각 인스턴스별 metricLatest fetch
              └── 라벨 텍스트 업데이트
```

이 구조 때문에 HeatmapMixin의 `refreshInterval: 0`으로 설정합니다 (Mixin 자체 타이머 비활성).

3D 데이터 라벨 (CSS2DObject)

ActionPanel은 습도 모드에서 각 3D 컴포넌트 위에 데이터 라벨을 표시합니다.

```
// 라벨 생성
const labelDiv = document.createElement('div');
labelDiv.className = 'data-label';
labelDiv.textContent = '25.3° C';

const css2dObject = new THREE.CSS2DObject(labelDiv);

// 3D 오브젝트 바운딩 박스 위에 위치
const box = new THREE.Box3().setFromObject(inst.appendElement);
const center = new THREE.Vector3();
box.getCenter(center);
css2dObject.position.copy(center);

// 인스턴스의 3D 오브젝트에 추가
inst.appendElement.add(css2dObject);

// 추적 배열에 저장 (cleanup용)
this._dataLabels.push({ instance: inst, css2dObject });
```

---

5. 페이지 수준 인프라

페이지 스크립트 실행 순서

```
page_scripts/before_load.js    ← 이벤트 버스, 레이캐스팅 초기화
  ↓
[컴포넌트 register.js 실행]     ← 각 컴포넌트 초기화
  ↓
page_scripts/loaded.js         ← GlobalDataPublisher 등록, 초기 데이터 fetch
  ↓
[사용자 인터랙션]
  ↓
page_scripts/before_unload.js  ← 전체 cleanup
```

before_load.js

3D 레이캐스팅 이벤트 시스템을 초기화합니다.

```
// 3D 오브젝트 클릭 감지 → 이벤트 버스로 전파
Weventbus.on('@assetClicked', handler);
```

```
Weventbus.on('@assetSelected', handler);
```

이 이벤트를 컴포넌트의 `bind3DEvents(this, { click: '@assetClicked' })` 가 수신합니다.

loaded.js

페이지 수준 데이터 매핑을 등록합니다.

```
this.pageDataMappings = [
  { topic: 'assetList', datasetName: 'assetList', param: { ... } },
  { topic: 'relationList', datasetName: 'relationList', param: { ... } },
];

// GlobalDataPublisher에 등록
GlobalDataPublisher.registerMapping(this.pageDataMappings);
GlobalDataPublisher.fetchAndPublish('assetList', this);
```

개별 컴포넌트는 `fetchData(this.page, datasetName, param)` 으로 자체 데이터셋을 호출합니다.

before_unload.js

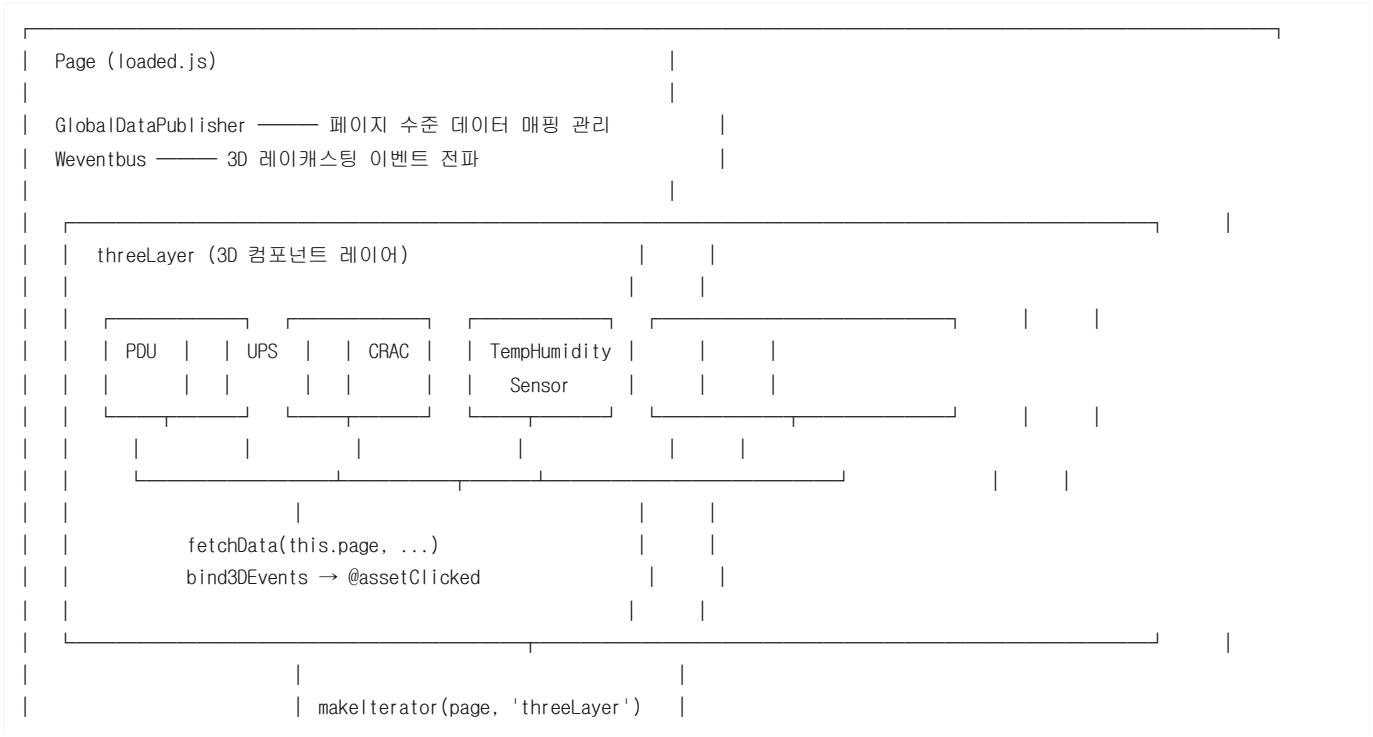
전체 리소스를 정리합니다.

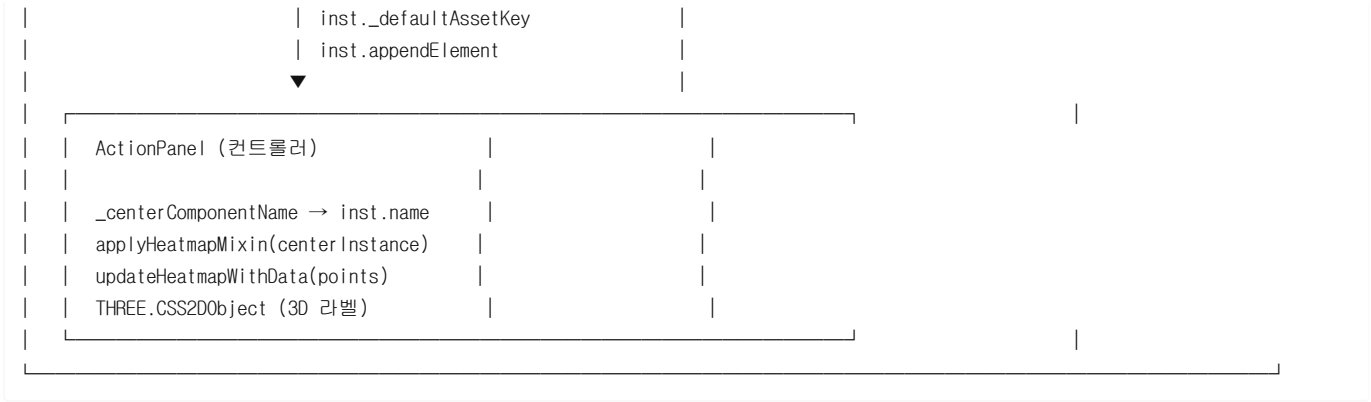
```
// GlobalDataPublisher 해제
GlobalDataPublisher.unregisterMapping(this.pageDataMappings);

// 이벤트 버스 해제
Weventbus.off('@assetClicked', handler);

// 3D 히트맵 인스턴스 정리
// 각 컴포넌트의 beforeDestroy.js에서 destroyPopup(), destroyHeatmap() 호출
```

---

6. 컴포넌트 간 통신 구조

통신 방식 비교





통신 패턴 요약

패턴	방향	메커니즘	예시
3D 클릭 → 컴포넌트	Page → Component	`Weventbus + bind3DEvents`	3D 오브젝트 클릭 → showDetail()
컴포넌트 → 데이터	Component → API	`fetchData(page, datasetName, param)`	자산 상세, 메트릭 조회
ActionPanel → 3D 컴포넌트	Component → Component	`makeIterator` + 직접 메서드 호출	toggleHeatmap(), updateHeatmapWithData()
컴포넌트 → 3D 씬	Component → Three.js	`wemb.threeElements` 접근	카메라 이동, 좌표 변환

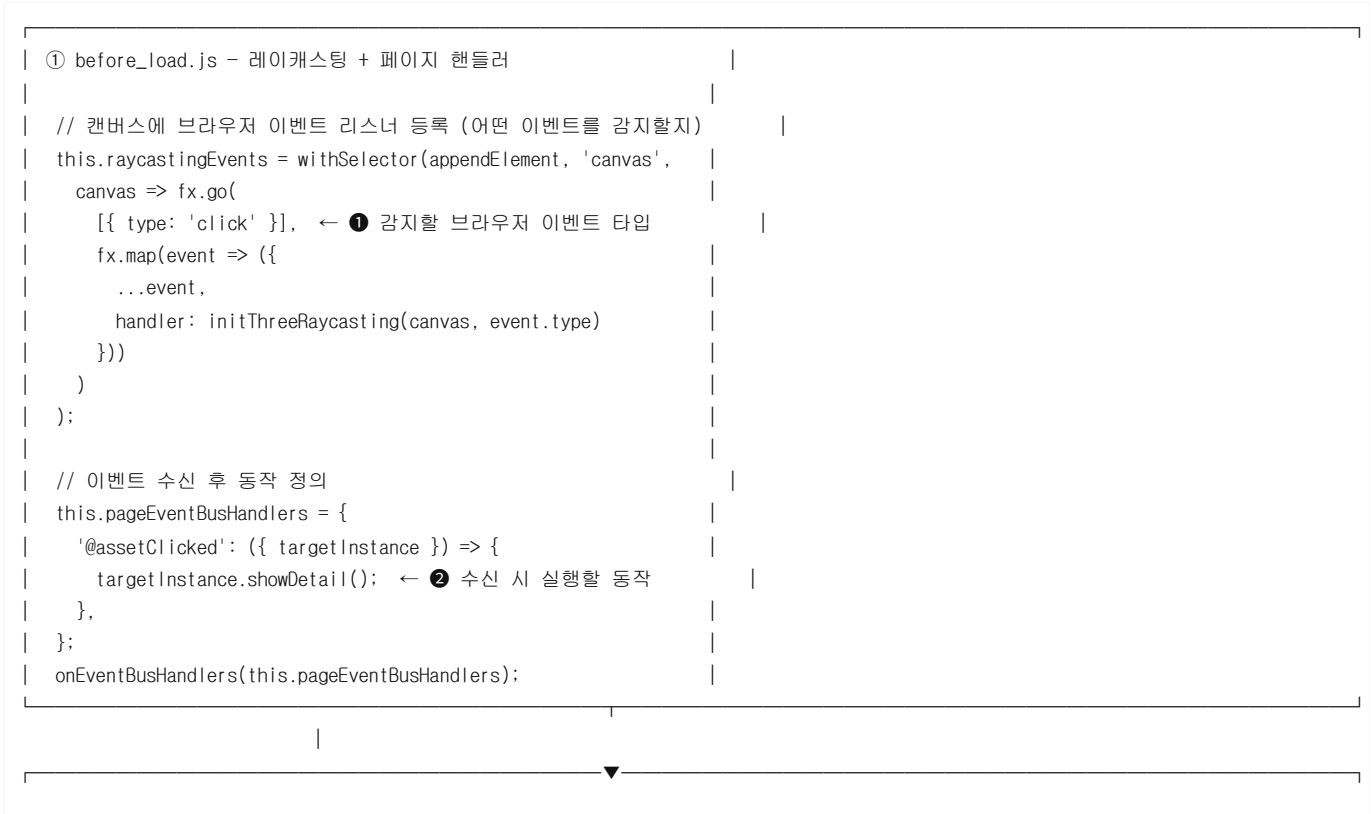
핵심: 컴포넌트 간 직접 참조는 ActionPanel만 수행합니다. 나머지 컴포넌트는 `this.page` 를 통한 데이터 패치와 이벤트 버스를 통한 이벤트 수신만 합니다.

---

7. 이벤트 시스템 현황 및 커스텀

3D 이벤트 전체 흐름

3D 컴포넌트에서 이벤트가 동작하려면 **3곳을 모두 설정**해야 합니다.



```
| ② 컴포넌트 register.js - customEvents 선언                    |
|                                                               |
|  this.customEvents = {                                        |
|    click: '@assetClicked',  ← ❸ 브라우저 이벤트 → 커스텀 이벤트명  |
|  };                                                           |
|  bind3DEvents(this, this.customEvents);                       |
└──────────────────────────┬────────────────────────────────────┘
                           |
┌──────────────────────────▼────────────────────────────────────┐
| ③ Wkit 내부 처리 (자동)                                        |
|                                                               |
|  makeRaycastingFn: 캔버스 클릭 → 레이캐스트 → 교차 오브젝트 탐색   |
|       ↓                                                       |
|  make3DHandler: customEvents[event.type] 조회                 |
|       ↓                                                       |
|  Weventbus.emit('@assetClicked', { event, targetInstance })   |
|       ↓                                                       |
|  pageEventBusHandlers['@assetClicked'] 실행                   |
|       ↓                                                       |
|  targetInstance.showDetail()                                  |
└───────────────────────────────────────────────────────────────┘
```

컴포넌트별 이벤트 현황

3D 장비 컴포넌트

컴포넌트	customEvents	팝업 내부 이벤트	Weventbus 발행
PDU	`{ click: '@assetClicked' }`	`.close-btn, .tab-btn`	없음
UPS	`{ click: '@assetClicked' }`	`.close-btn, .tab-btn`	없음
SWBD	`{ click: '@assetClicked' }`	`.close-btn, .tab-btn`	없음
CRAC	`{ click: '@assetClicked' }`	`.close-btn, .heatmap-btn`	없음
TempHumiditySensor	`{ click: '@assetClicked' }`	`.close-btn, .heatmap-btn`	없음

모든 3D 장비 컴포넌트는 동일한 패턴: `bind3DEvents` → `@assetClicked` → `showDetail()`

2D UI 컴포넌트

컴포넌트	customEvents	내부 핸들러	Weventbus 발행
ActionPanel	`{}` (비어있음)	`.action-panel` click → `handleTabSwitch`	없음
AssetList	`{ click: { '.refresh-btn': '@refreshClicked' } }`	검색, 필터, 트리 클릭	`@assetSelected`, `@assetNodeSelected`
AssetTree	`{ click: { '.refresh-btn': '@refreshClicked' } }`	트리 토글, 더블클릭(카메라 이동)	없음

ActionPanel의 handleTabSwitch (사용자가 선택한 line 112):

```
// ActionPanel은 Weventbus를 쓰지 않고 직접 DOM 이벤트를 처리합니다
this._internalHandlers = {
  btnClick: function (e) {
    const btn = e.target.closest('.action-btn');
    if (!btn) return;
    const action = btn.dataset.action;  // 'humidity' 또는 'temperature'
    handleTabSwitch.call(ctx, action);   // 내부 상태 전환
```

```
  },
};

panel.addEventListener('click', this._internalHandlers.btnClick);
```

페이지가 알 필요 없는 내부 UI 동작이므로 `_internalHandlers` 패턴을 사용합니다. (판단 기준: EVENT_HANDLING.md)

이벤트 커스텀 방법

예시 1: 클릭 → 더블클릭으로 변경

팝업을 더블클릭으로 열도록 변경하려면 **3곳을 모두 수정**해야 합니다.

① before_load.js — 레이캐스팅에 dblclick 추가:

```
// 변경 전
[{ type: 'click' }]

// 변경 후
[{ type: 'dblclick' }]
// 또는 click과 dblclick 모두 감지
[{ type: 'click' }, { type: 'dblclick' }]
```

② before_load.js — 핸들러 이벤트명 변경:

```
// 변경 전
this.pageEventBusHandlers = {
  '@assetClicked': ({ targetInstance }) => {
    targetInstance.showDetail();
  },
};

// 변경 후
this.pageEventBusHandlers = {
  '@assetDblClicked': ({ targetInstance }) => {
    targetInstance.showDetail();
  },
};
```

③ 컴포넌트 register.js — customEvents 변경:

```
// 변경 전
this.customEvents = {
  click: '@assetClicked',
};

// 변경 후
this.customEvents = {
  dblclick: '@assetDblClicked',
};
```

before_unload.js는 수정 불필요 — `clearRaycasting()` 이 `this.raycastingEvents` 를 순회하며 모든 타입을 자동 정리합니다.

예시 2: 클릭과 더블클릭에 서로 다른 동작 할당

```
// ① before_load.js - 두 이벤트 타입 모두 레이캐스팅 등록
this.raycastingEvents = withSelector(this.appendElement, 'canvas', canvas =>
  fx.go(
```

```
    [{ type: 'click' }, { type: 'dblclick' }],  // 두 이벤트 모두
    fx.map(event => ({
      ...event,
      handler: initThreeRaycasting(canvas, event.type)
    }))
  )
);

// ① before_load.js - 각각 다른 핸들러 등록
this.pageEventBusHandlers = {
  '@assetClicked': ({ targetInstance }) => {
    targetInstance.showDetail();      // 클릭 → 팝업
  },
  '@assetFocused': ({ targetInstance }) => {
    targetInstance.zoomToFit?.();     // 더블클릭 → 카메라 이동
  },
};

// ③ 컴포넌트 register.js - 이벤트 타입별 커스텀 이벤트명
this.customEvents = {
  click: '@assetClicked',
  dblclick: '@assetFocused',
};
bind3DEvents(this, this.customEvents);
```

예시 3: 2D 컴포넌트에 새 이벤트 추가

2D 컴포넌트는 페이지 스크립트 수정 없이 `bindEvents` 만으로 가능합니다.

```
// 컴포넌트 register.js
this.customEvents = {
  click: {
    '.refresh-btn': '@refreshClicked',
    '.export-btn': '@exportRequested',  // ← 새 이벤트 추가
  },
  change: {
    '.filter-select': '@filterChanged', // ← 다른 이벤트 타입도 가능
  },
};
bindEvents(this, this.customEvents);
```

페이지에서 수신하려면 `before_load.js` 에 핸들러를 추가합니다:

```
this.pageEventBusHandlers = {
  // ... 기존 핸들러 ...
  '@exportRequested': ({ targetInstance }) => {
    console.log('Export from:', targetInstance.name);
  },
};
```

3D 이벤트 vs 2D 이벤트 비교

구분	3D (bind3DEvents)	2D (bindEvents)
대상	Three.js Object3D (appendElement)	DOM 요소 (appendElement)
이벤트 감지	레이캐스팅 (before_load.js 필수)	DOM 이벤트 위임 (자체 완결)

customEvents 형식	`{ click: '@eventName' }`	`{ click: { '.selector': '@eventName' } }`
셀렉터	없음 (3D 오브젝트 전체)	CSS 셀렉터로 특정 요소
페이지 스크립트 필요	**필수** (initThreeRaycasting)	**선택** (핸들러 등록 시만)
cleanup	dispose3DTree에서 eventListener 제거	removeCustomEvents로 제거

핵심 차이: 3D 이벤트는 **반드시 페이지 스크립트(before_load.js)에서 레이캐스팅을 설정**해야 동작합니다. 컴포넌트의 `bind3DEvents` 만으로는 이벤트가 발생하지 않습니다.

내부 핸들러 vs customEvents 판단 기준

> *자세한 원칙은 EVENT_HANDLING.md 참조*

```
"이 동작의 결과를 페이지가 알아야 하는가?"

  예 → customEvents + bindEvents/bind3DEvents
       → Weventbus로 발행 → 페이지 핸들러에서 수신

  아니오 → _internalHandlers + addEventListener
           → 컴포넌트 내부에서만 처리
```

예시	페이지가 알아야?	방식
3D 클릭 → 팝업 열기	O	`customEvents` → `@assetClicked`
팝업 내 탭 전환	X	`bindPopupEvents` (Shadow DOM 내부)
ActionPanel 버튼 토글	X	`_internalHandlers`
AssetList 행 선택	O	`Weventbus.emit('@assetSelected')`
검색 입력 필터링	X	`_internalHandlers`

관련 문서

문서	내용
MIXIN_TUTORIAL.md	PopupMixin/HeatmapMixin step-by-step 튜토리얼
POPUP_MIXIN_API.md	PopupMixin 메서드 레퍼런스
HEATMAP_MIXIN_API.md	HeatmapMixin 옵션 레퍼런스
WKIT_API.md	bind3DEvents, fetchData, makeIterator
EVENT_HANDLING.md	이벤트 처리 원칙 (customEvents vs _internalHandlers)
WEVENTBUS_API.md	Weventbus 이벤트 버스 API
GLOBAL_DATA_PUBLISHER_API.md	GlobalDataPublisher

최종 업데이트: 2026-02-25